J Intell Inf Syst (2018) 50:129–164
DOI 10.1007/s10844-017-0445-8

Discrete representation strategies for foreign exchange prediction

Blaž Žličar[1] · **Simon Cousins**[1]

Received: 2 July 2016 / Revised: 14 November 2016 / Accepted: 13 January 2017 /
Published online: 11 February 2017

Abstract *This is an extended version of the paper presented at the* 4*th International Workshop NFMCP 2015 held in conjunction with ECML PKDD 2015. The initial version has been published in NFMCP 2015 conference proceedings as part of Springer Series.* This paper presents a novel approach to financial times series (FTS) prediction by mapping hourly foreign exchange data to string representations and deriving simple trading strategies from them. To measure the degree of similarity in these *market strings* we apply familiar string kernels, bag of words and n-grams, whilst also introducing a new kernel, time-decay n-grams, that captures the temporal nature of FTS. In the process we propose a sequential Parzen windows algorithm based on discrete representations where trading decisions for each string are learned in an online manner and are thus subject to temporal fluctuations. We evaluate the strength of a number of representations using both the string version and its continuous counterpart, whilst also comparing the performance of different learning algorithms on these representations, namely support vector machines, Parzen windows and Fisher discriminant analysis. Our extensive experiments show that the simple string representation coupled with the sequential Parzen windows approach is capable of outperforming other more exotic approaches, supporting the idea that when it comes to working in high noise environments often the simplest approach is the most effective.

Keywords Time series prediction · Discretisation · Parzen windows · String kernels · Support vector machines · Fisher discriminant analysis

✉ Blaž Žličar
b.zlicar@cs.ucl.ac.uk

Simon Cousins
s.cousins@cs.ucl.ac.uk

[1] University College London, London WC1E 6BT, UK

1 Introduction

Financial time series (FTS) are renowned for being extremely noisy thus making it difficult to make predictions regarding future events based upon these observations. This has led many previous authors to seek out novel and exotic representations of the time series that attempt to remove some of this noise, which would allow for better predictions to be made. In this paper we adopt an alternative approach to the problem: that is, the mapping of hourly foreign exchange rate data to a string representation, and using this string to derive a simple trading strategy. In the process we propose a new type of string kernel, i.e. time-decay n-grams, and a sequential Parzen windows algorithm based on discrete representations.

We begin our representation by constructing an alphabet using an arbitrary partitioning of the real-valued returns of the currency pair, and map each hourly return to a particular letter in the alphabet. By concatenating the letters associated with the previous hourly returns we create a *market string*, which is assumed to be representative of the current market conditions, and is used to guide trading decisions. Throughout the paper we show how kernel methods can be used to extend the simple string representation to capture different characteristics of the financial time series. Furthermore, we present a new temporal-based string kernel, i.e. time-decay n-grams, that takes into consideration how recently a particular substring pattern occurred. As well as evaluating the strength of the time series representation through strings, we also investigate the impact of creating a predictor using a particular algorithm. Namely, we compare the performance of Parzen windows (PW), support vector machines (SVMs), Fisher discriminant analysis (FDA) and our incremental learning algorithm similar to a Parzen windows. All of these algorithms and representations are tested on almost 8 years worth of hourly data taken from four of the most actively traded currency pairs: AUD/USD, CHF/USD, EUR/USD and GBP/USD. The results seems to favour the view that simple approaches tend to work best when working in noisy environments. The paper continues with a review of previous work in Section 2, which is followed by a short introduction to kernels in Section 3, paying particular attention to the string-based ones used in our experiments. In Sections 4 and 5 we describe the method of constructing the so called *market alphabet* as well as batch algorithms used to define trading strategies. Section 6 then outlines the sequential Parzen windows algorithm based on discrete representations. We conduct the experiments in Sections 7, 8 and 9 and finish with conclusions and advise on future directions of research in Section 10.

2 Previous work

This paper addresses the problem of FTS prediction from two aspects: the first concerns how we present the data to the algorithm and the second concerns the choice of algorithm used to construct the predictor. A large part of FTS research focuses on the discovery of patterns from noisy data. The traditional approach to model selection usually involves some form of autoregressive model that we have to fit a set of parameters in order to predict future price trajectories. While this approach has proven quite useful for the prediction of mean-reverting and persistent processes such as volatility, it is less successful in predicting the value or even directional movement of prices.

Machine learning has demonstrated a number of encouraging results for the prediction of FTS, with SVMs being one of the most popular approaches. One of the first applications of SVMs to FTS was presented in Tay and Cao (2001) and later extended in Kim (2003). Both papers provide an empirical analysis of SVM-based FTS prediction and compare its performance against a number of other techniques including multi-layer back-propagation neural network and case-based reasoning. The experimental results in both papers suggest a superiority of SVM methods when compared to similar techniques, however they do outline the challenges of the SVM approach in terms of generalisation to unseen data. In Van Gestel et al. (2001) the authors compare the performance of a least squares SVM[1] to that of several autoregressive models as well as nonparametric models for both return and volatility prediction. They report that the least squares SVM has superior performance both in terms of achieving higher directional prediction and better overall performance compared to that of other models used in their investigation. Further applications of SVMs for financial forecasting can be found in Cao and Tay (2003), Perez-Cruz et al. (2003), Hossain and Nasser (2011), Ou and Wang (2010), Khan (2011). The other two algorithms applied in this paper, namely FDA and PW are much less explored in terms of FTS prediction. In terms of application in financial space, FDA has been mainly investigated as a bankruptcy prediction classifier and PW most frequently used for nonparametric density estimation. A recent application of PW to prediction of stock prices in U.S. and U.K. markets that reportedly showed encouraging results is conducted in Mwamba (2011).

In terms of data representation, the prevalent approach is to work with continuous time series data, favouring returns instead of prices due to a number of statistical properties that the former possess, such as weak stationarity. Relatively less attention has been given to the study of the discretisation of FTS, nonetheless there have been several attempts to reduce the noise of a time series by using various quantisation techniques as well as symbolic representations. The latter include discretisation methods and string representations applied in this paper (see for example Lin et al. 2003), although their application for prediction of FTS in particular and for the purpose of tactical asset allocation is explored to a much lesser extent. A concise and informative overview of time series data mining techniques can be found in Fu (2011). Therefore, we focus less on the alphabet construction method and pay more attention to the way patterns are presented to an algorithm, namely we introduce a new type of string kernel aimed specifically at FTS that are known for their time-conditional characteristics. A popular approach to FTS prediction among practitioners is to employ rule-based prediction methods that allow the incorporation of prior knowledge into the decision-making process. These rule-based prediction methods involve two sources of knowledge; the first is forecasting expertise (e.g. quantitative extrapolations and modelling) and the second is domain knowledge (practical knowledge about causal relations within particular field) (Armstrong et al. 2001). Perhaps the most popular example of the latter in finance are methods where rules are based on technical indicators.[2] These allow a researcher to include their expert knowledge into the forecasting process in the form of various thresholds and patterns applied to technical indicators which ultimately leads to a discrete evaluation of the market at a specific point in time. Understandably, rule-based forecasting goes beyond rules based on technical indicators, e.g. assigning different considerations to level and trend of a time series, combining predictions of a number of models, separating models according to their forecast horizons etc. (Armstrong et al. 2001). In this sense, machine learning is less restrictive compared to classical time series analysis since it does not necessarily require that a time series satisfies a specific set of assumptions. While econometrics normally avoids including technical

[1] An extension of the original SVM that penalises the slack variables according to their squared value.

[2] Technical indicators are best described as rule-based evaluations of the underlying time series where their mathematical formulae is not based on statistical theory but on an expert's domain knowledge instead.

indicators and other heuristics, machine learning techniques allow for an easy inclusion of such indicators without violating any statistical assumptions. In fact it is quite popular to use these technical indicators as building blocks of the feature space when predicting FTS with machine learning algorithms. Moreover, it extends the possibilities of data representation through the use of kernel functions. The majority of implementations use the Gaussian kernel function due to its rich representation ability, however linear and polynomial kernels are also popular choices. To the best of our knowledge, this paper presents the first use of string (text) kernels for prediction of FTS. More importantly, we introduce a new type of string kernel, namely the time-decay n-grams, that is constructed so as to capture the market dynamics and domain knowledge for a specific type of data. As the use of kernels plays an important role in this paper we provide a short overview of kernel methods in the following section. For more details on kernel methods we point the interested reader to Shawe-Taylor and Cristianini (2004) and Schölkopf and Smola (2002).

3 Kernel methods

Suppose we are given a set of training examples $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \mathcal{Y}$. The general idea of pattern analysis is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps input observations $x \in \mathcal{X}$ to output values $y \in \mathcal{Y}$. In this paper we focus on binary classification algorithms where the output space is given by $\mathcal{Y} = \{-1, 1\}$ and the input space $\mathcal{X}$ is arbitrary. When using training samples to learn the function $f : \mathcal{X} \rightarrow \mathcal{Y}$, we want to ensure that it is capable of generalising to unseen examples. Therefore given a new input x we want to predict the corresponding $y \in \mathcal{Y}$, such that the input-output pair (x, y) is similar to what we have seen in our training sample. In order to do this we must define a similarity measure over both the input and output space. In classification, output similarities are simply given by whether they belong to the same class. In the kernel method approach to learning, we define the similarity measure on the input space by introducing a kernel function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. In order to be a valid similarity measure, the kernel function must satisfy that for all $x, z \in \mathcal{X}$

$$k(x, z) = \langle \phi(x), \phi(z) \rangle,$$

where ϕ, known as the feature mapping, maps $\mathcal{X}$ into some dot product space $\mathcal{F}$ where similarities can be evaluated. This very general notion of a kernel function allows the input space to be mapped to a large number of possible feature spaces, often of much higher dimensions, where the differences between observations drawn from different classes may become more distinct. A benefit of the algorithms using kernel methods is that they avoid having to explicitly compute feature space representations by ensuring that feature mappings only appear in the algorithms as inner-products and can thus be directly evaluated using a kernel function.

Another point worth stressing is that our definition of a feature mapping, and therefore the kernel function, represents our prior knowledge about the research problem and is a crucial stage of the pattern analysis. Kernel functions act as similarity measures and offer flexibility in terms of incorporating domain knowledge into the pattern analysis task. In this paper we attempt to capture some domain knowledge about the intra-day FX price movements through the discretisation of the returns of the FTS and through introduction of time-decay n-grams so as to capture temporal information of the FTS. Loosely speaking, we conjecture that when traders view the hourly changes of an FTS they are unlikely to take into consideration the exact value of the hourly returns (the continuous value). Instead we believe that it is more likely that they conduct some form of discretisation of the previous price movements in order to gauge the underlying market conditions, and make their trading decisions. This is in essence what we are attempting to do by creating the *market alphabet*, and concatenating previous hourly returns and their corresponding letters to form a string representation of the market. We now present three different string kernels used in our experiments, each of which is capable of capturing different characteristics of the time series. First we outline a couple of well-known text kernels, namely bag-of-words and *n*-grams. Next, we propose a new type of string kernel, the time-decay *n*-grams, aimed at capturing conditional nature of FTS.

3.1 Bag-of-words

Bag-of-words (BoW) approach is a type of vector space model where the idea is to count the number of words appearing in a document and make a classification decision based on that information. BoW does not take into account order of words so grammatical or semantic information is lost to a high degree. As an example, let us assume a set of documents L on two different topics A and B, i.e. we have two classes of documents. The BoW approach makes a vector representation of each document by counting the number of times a particular word appears in that document where the dimensionality of the vector is determined by the dimensionality M of the dictionary (union of all words in all documents). A mapping of a document d_l into a vector space can be written as

$$\phi : d_l \rightarrow \phi(d_l) = (f(w_1, d_l), \ldots, f(w_m, d_l)) \in \mathbb{R}^M \tag{1}$$

where $f(w_m, d_l)$ is the frequency of word w_m in a document d_l. After performing identical mapping on all sets of documents we can then store the data in a document-term matrix $\mathbf{D}$ of dimensions $L \times M$ with rows associating to the number of documents and columns referring to the number of words. Our kernel matrix that presents the basis for a further pattern analysis is then $\mathbf{K} = \mathbf{D}\mathbf{D}^{\intercal}$. Note that within the context of our research, a *document* is equivalent to a single word of length K and hence we are effectively comparing entire documents when estimating the degree of similarity between words. Using string kernel terminology, BoW approach is equivalent to fixed-length strings matching. Let us follow Shawe-Taylor and Cristianini (2004) and define an alphabet Σ as a finite set of $|\Sigma|$ letters and a string $s = s_1, \ldots, s_{|s|}$ as any finite sequence of letters of some predefined length $|s| = K$ from Σ with the set of all finite strings equal to Σ^K. BoW is then equal to a boolean function $p(s, s')$ that returns a value of 1 only when the two strings, s and s', are identical and 0 otherwise:

$$p(s, s') = \begin{cases} 1 & \text{if} \quad s = s' \\ 0 & \text{if} \quad otherwise \end{cases} \tag{2}$$

As such BoW kernel does not include finer comparison of words such as taking into account subsequences of characters within words. A string kernel presented in the following section does precisely that.

3.2 *n*-grams

This is a text representation method popular especially in computational linguistics (Lodhi et al. 2002) and bioinformatics (Liu et al. 2008). A document is represented in terms of substrings where each substring represents a feature of the underlying document. As its name suggests n-grams (NG) refers to n-number of adjacent characters in the alphabet with each n-gram type representing a type of substring (i.e. a feature). Within the context of our research where documents are in fact represented by a single word, we now no longer look for similarities between strings (words) of fixed length K, but also take into account similarities between subsequences of length $n \leq K$ within each word. The dimensionality of our feature space is now $|\Sigma|^n$, where $n \leq K$ with n denoting the length of subsequences, i.e. n-grams. Note that the similarity function for each string is no longer a boolean operator and our feature space no longer comprised of binary vectors. By accounting for subsequences we now have a feature space where each string is represented by a vector of non-zero entries for each of the n-grams present in that string. We now present a simple example of the n-gram kernel computations on a set of four five-letter strings ($K = 5$) constructed from a three-letter alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ where the length of the subsequence that we are interested in is $n = 2$.

Example 1 Consider the strings abcab, abbbc, cbaba and bbcaa. We fix the string subsequence length to $n = 2$ and compute feature space representation for each of these words. To do this we compute all contiguous subsequences of length $n = 2$ present in the data set. We then identify whether this subsequence was present in a particular word so as to obtain a feature vector for each string:

ϕ	ab	bc	ca	bb	cb	ba	aa
abcab	2	1	1	0	0	0	0
abbbc	1	1	0	2	0	0	0
cbaba	1	0	0	0	1	2	0
bbcaa	0	1	1	1	0	0	1

Finally we quantify the similarities between the four words by calculating the kernel matrix:

K	abcab	abbbc	cbaba	bbcaa
abcab	6	3	2	2
abbbc	3	6	1	3
cbaba	2	1	6	0
bbcaa	2	3	0	4

Note, that the kernel matrix assigns the highest values along the diagonal where the words are in fact identical, while having non-zero non-diagonal entries as a result of taking into account the subsequences as part of string comparison.

3.3 Time-decay n-grams

We now introduce a time-decay n-gram kernel that takes into account the time-conditional information of an FTS. We attempt to do so by assigning higher weights to the more recent FTS observations compared to less recent ones. If we revisit the traditional n-grams in Example 1 and assume that these letters correspond to a time series, we can incorporate the temporal nature of the observations by introducing the weights q_i for $i = \{1, \ldots, 4\}$. Let i index the location of the n-gram, where $i < j$ means i occurred after j and we assume that

$\sum_i q_i = 1$ and $q_i \geq q_j$ if $i \leq j$. For example, suppose we wanted the relationship between the first n-gram and the last one to be such that $q_1/q_4 = d$, meaning the first n-gram is d times more important than the last one. This can be achieved through a simple exponential decay function were the ratio between the importance of consecutive n-grams is fixed, i.e. $q_i/q_{i+1} = \Delta = \exp(\log(d)/(k-1))$. Here k denotes the maximum number of n-grams for a given word length K and is given as $k = \max(K - n + 1, 0)$. In our case with $K = 5$ and $n = 2$ the maximum number of n-grams is therefore $k = 4$.

Example 2 Consider once again the strings abcab, abbbc, cbaba and bbcaa. We fix the string subsequence length to $n = 2$ and compute feature space representation for each of these words. To do this we compute all contiguous subsequences of length two present in the data set. We then identify whether this subsequence was present in a particular word so as to obtain a feature vector for each string, and assign it a weight q_i corresponding to its position in the string:

ϕ	ab	bc	ca	bb	cb	ba	aa
abcab	$(q_1 + q_4)$	q_2	q_3	0	0	0	0
abbbc	q_1	q_4	0	$(q_2 + q_3)$	0	0	0
cbaba	q_3	0	0	0	q_1	$(q_2 + q_4)$	0
bbcaa	0	q_2	q_3	q_1	0	0	q_4

Suppose we set the decay factor at $d = 2$ and therefore $q_i/q_{i+1} = \Delta = \exp(\log(2)/(3))$. Then the kernel matrix between these words is given by:

K	abcab	abbbc	cbaba	bbcaa
abcab	0.38	0.22	0.11	0.12
abbbc	0.22	0.38	0.07	0.21
cbaba	0.11	0.07	0.36	0.00
bbcaa	0.12	0.21	0.00	0.27

4 Discrete time series representations

Here we demonstrate a method for recoding of continuous data into a time series of discrete representations, i.e. a so called *market alphabet* that contains information about the market dynamics for the underlying time period. Generally speaking, information about market action for a particular asset is obtained by recording four prices at each time interval. In our case of hourly data, we have prices for the open, high, low and close over each hourly interval, given by O_t, H_t, L_t and C_t respectively, with each price being a positive real number. As will be made clear in subsequent sections, our goal will be to predict at the beginning of the price interval whether the price will increase or decrease over the interval based on recent relative price movements, or more precisely simple arithmetic returns, $r_t = (C_t - O_t)/O_t$ that have occurred leading up to time t. Rather than following the usual practice of using the continuous values of the returns we form a partitioning of the real line $\mathbb{R}$ and label each sub-interval with a unique identifier, i.e. a letter σ from alphabet Σ.

The approach we take is to use sub-intervals $(b_k, b_{k+1}]$, where $b_k < b_{k+1}$, that have roughly an equal number of members, i.e. in the training sample there are roughly the same number of returns r_t corresponding to each sub-interval. This can be achieved by sorting the returns r_t and taking equally separated percentile points as the limits of the sub-intervals. An important point to consider is the sub-interval that contains both positive and negative values. Intuitively it makes sense to split this sub-interval into two separate sub-intervals either side of zero. The mapping $A : \mathbb{R} \rightarrow \varSigma$ between returns and the alphabet is given by

$$A(r) = \begin{cases} \sigma_1, & b_0 < r \leq b_1 \\ \sigma_2, & b_1 < r \leq b_2 \\ \quad\vdots & \\ \sigma_{|\varSigma|}, & b_{|\varSigma|-1} < r \leq b_{|\varSigma|}, \end{cases} \tag{3}$$

where $b_0 = -\infty$ and $b_{|\varSigma|} = +\infty$. Using this notion each observation r_t belongs to a given sub-interval, which we identify with a letter σ from our alphabet $\varSigma$. The representation can be extended to include past observations by concatenating the letters corresponding to past returns (see Fig. 1). More formally, an alphabet $\varSigma$ of $|\varSigma|$ letters is constructed by an arbitrary partitioning of the real line $\mathbb{R}$ where each return $r_t \in \mathbb{R}$ is mapped to a letter $\sigma_t \in \varSigma$. A string is constructed according to $s_t = \sigma_{t-1} \ldots \sigma_{t-K}$, where K defines how many past returns we look at.

5 Batch classification strategies

As mentioned earlier, the goal is to predict at the beginning of the price interval whether the price will increase or decrease over the interval, i.e. $y_t = \text{sign}\,(C_t - O_t) \in \{-1, 1\}$. Hence,

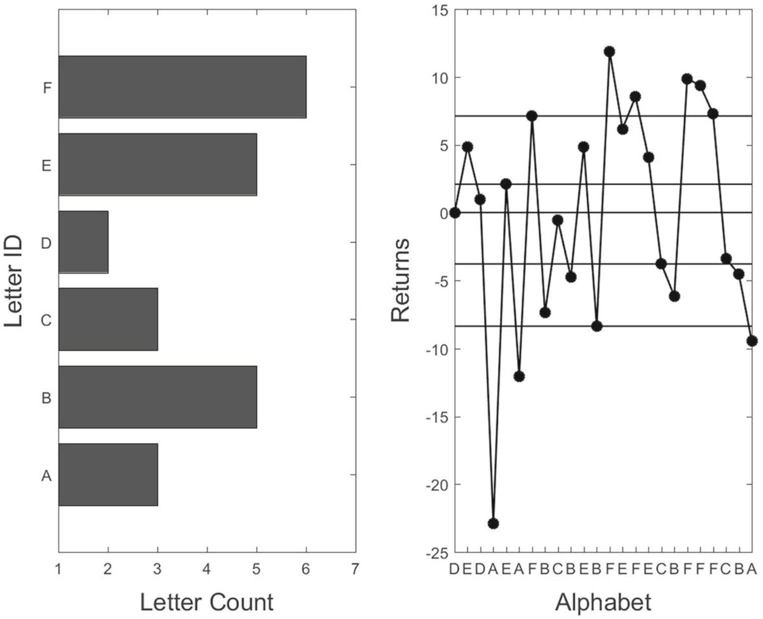


Fig. 1 Alphabet representation: *left side* displays letters (i.e. partitions) together with the number of examples (namely, the returns of EUR/USD exchange rate) that fall into individual partitions. *Right side* displays the time series of these same hourly returns over the period of 24 h with each of the hourly returns being assigned a letter depending on which partition they fall into

we will employ binary classification algorithms that return predictions $\hat{y}_t \in \{-1, 1\}$ based on which we then buy (go *long*) or sell (go *short*) the underlying currency cross. Classifiers investigated here and presented in the following sections are Parzen windows (PW), support vector machines (SVM) and Fisher discriminant analysis (FDA).

5.1 Parzen windows

We begin by describing a nonparametric conditional probability estimator proposed originally by Nadaraya (1964) and Watson (1964) with the following description borrowed from Smola (2007). Suppose we have a training set of observations $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_t, y_t)\} \in \mathcal{X} \times \mathcal{Y}$ and wish to estimate a conditional probability $P(y|\mathbf{x})$. For the moment let us assume that target takes values $y_t \in \{-1, 1\}$. What is also known as Watson-Nadaraya (WN) model, estimates conditional probability by combining the product rule $p(\mathbf{x}, y) = p(y|\mathbf{x})p(\mathbf{x})$ and Bayes rule, so that

$$P(y|\mathbf{x}) = \frac{P(y, \mathbf{x})}{P(\mathbf{x})} = \frac{\sum_t y_t k(\mathbf{x}_t, \mathbf{x})}{\sum_t k(\mathbf{x}_t, \mathbf{x})}, \tag{4}$$

where we estimate both densities $p(y, \mathbf{x})$ and $p(\mathbf{x})$ using Parzen windows. Here $k(\mathbf{x}_t, \mathbf{x})$ denotes a kernel function that satisfies $\sum_t k(\mathbf{x}_t, \mathbf{x}) = 1$ and acts as a weighting function that defines the region of influence of a data point. Optimal decision rule for WN classifier is then equal to observing the sign of

$$P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x}) \propto \sum_t y_t k(\mathbf{x}_t, \mathbf{x}). \tag{5}$$

Note, that we can easilty turn WN classifier into regression by simply changing the target from a categorical $y_t \in \{-1, 1\}$ to real-valued numeric output $y_t \in \mathcal{R}$. Parzen windows estimators are not new in financial analysis, where they are usually applied as empirical density estimators. However, due to their nonparametric nature they are often dismissed due to risk of overfitting.

5.2 Fisher discriminant analysis (FDA)

Fisher discriminant analysis (Fisher 1936) is an approach to classification that uses only the empirical mean and covariance matrices of the class specific distributions to construct a predictor. Suppose we have the empirical mean $\hat{\mu}_i$ and covariance matrices $\hat{\Sigma}_i$ for each class $i = -, +$. The goal of FDA is to find the vector $\mathbf{w}$ that maximises the distance between the projected means, whilst ensuring that the within-class variance remains small. Mathematically speaking, this is represented by maximising the functional

$$J(\mathbf{w}) = \frac{\left(\mathbf{w}^T(\hat{\mu}_+ - \hat{\mu}_-)\right)^2}{\mathbf{w}^T\left(\hat{\Sigma}_+ + \hat{\Sigma}_-\right)\mathbf{w}},$$

and it is well known that the optimal solution $\mathbf{w}^*$ is given by

$$\mathbf{w}^* = \left(\hat{\Sigma}_+ + \hat{\Sigma}_-\right)^{-1}(\hat{\mu}_+ - \hat{\mu}_-).$$

FDA represents one of the earliest approaches to pattern recognition, however it was relatively overlooked as a method for high-dimensional problems due to the difficulties of inverting a sometimes singular matrix. However, following the kernelisation of FDA in Mika et al. (1999), and its competitive performance with methods such as SVM and neural networks, interest was subsequently revived.

5.3 Support vector machines

Support vector machines (SVMs) (Cortes and Vapnik 1995; Shawe-Taylor and Cristianini 2000; Boser et al. 1992) are one of the most popular classifiers used within the machine learning community. Their popularity stems from the strong theoretical grounds upon which they were built and the generalisation performance that they offer. They were first introduced by Boser et al. (1992) as an algorithm for finding the optimal separating hyperplane for two classes and extended to account for the case of non-separable data by Cortes and Vapnik (1995). The SVM is a linear classifier operating in a high-dimensional space defined by the feature mapping $\phi(\mathbf{x})$ and they use the so called *kernel trick* to enable a dual representation and an efficient computation of the solution. The SVM is a maximum margin classifier where the margin is defined as the minimum distance of a training sample to the hyperplane defined by weight vector $\mathbf{w}$ and bias term b:

$$\gamma(\mathbf{w}, \mathcal{S}) = \min_i \frac{y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)}{||\mathbf{w}||} \tag{6}$$

It is the goal of the SVM algorithm to find the hyperplane defined by $\mathbf{w}$ and b such that this margin is maximised. Statistical learning theory (Vapnik 2013) states that the generalisation ability of a classifier depends on its VC-dimension and it was shown that this can be bounded in terms of the margin it produces. Therefore by maximising the margin of the classifier we can effectively bound the generalisation error of the classifier and it is this idea that has driven the theory behind SVMs. The slack-version of the optimisation (Cortes and Vapnik 1995) problem is given by:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b) \geq 1 - \xi_i \;\; i = 1 \dots n \\ & \xi_i \geq 0 \quad i = 1 \dots n. \end{aligned}$$

By using the standard Lagrangian formulation it is straightforward to show that the optimal weight vector $\mathbf{w}^*$ is given by a linear combination of the training points

$$\mathbf{w}^* = \sum_i \alpha_i y_i \phi(x_i),$$

where the conditions for optimality state that $\alpha_i > 0$ if $y_i(\mathbf{w}^T\phi(x_i) + b) \leq 1$, and $\alpha_i = 0$ if $y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1$. The optimal value for the bias term $b^* \in \mathbb{R}$ can also be extracted from the conditions for optimality, where for any j where $\alpha_j \in (0, C)$

$$b^* = y - j - \mathbf{w}^T\phi(x_j) = y_j - \sum_i \alpha_i y_i k(x_i, x_j).$$

Future predictions are made according to:

$$\hat{y} = \begin{cases} 1 & \text{if} \sum_i \alpha_i y_i k(x_i, x) + b \geq 0 \\ -1 & \text{otherwise} \end{cases}.$$

We will be using the formulation of SVM that uses ν parameter to control the degree of flexibility in SVM and effectively determines the lower bound on the number of support vectors.[3] Also, note that contrary to PW and FDA classifiers that base their predictions on

[3] Not to be confused with the ν parameter governing confidence of our prediction in sequential string algorithms presented in the following sections.

centres of the distribution, SVM constructs a predictor using the set of support vectors that effectively lie on the edge of the class specific distributions.

6 Discrete representation strategies

The following section presents three original sequential predictors that utilise discretisation of FTS and string representations, namely the (a) simple strings strategy (S-BoW), (b) a string subsequence strategy with equally-weighted n-grams (S-NG) and (c) a string subsequence strategy with time-decay n-grams (S-TDNG).

6.1 Simple strings strategy (S-BoW)

Our goal is to come up with a prediction rule $g : \Sigma^K \rightarrow \{-1, 1\}$ indicating whether we believe the price will increase or decrease over the next interval. Keeping in mind the discrete representation method described in Section 4, we know that the feature space ϕ is constructed by mapping each unique string $s \in \Sigma^K$ to binary vector with one non-zero entry corresponding to that particular string such that $\langle \phi(s), \phi(s') \rangle = 1$ if and only if $s = s'$, and zero otherwise. This is equivalent to a bag-of-words kernel presented earlier, where we only have a single word in each document. Here however, we learn our predictor in an incremental manner by maintaining an individual weight w_s and count c_s for each string. This weight w_s is given by the sum of observed outcomes for that string, i.e. the weight corresponding to string s at time T is given by $w_s = \sum_{t=1}^{T} y_t I[s_t = s]$, where $I[a]$ is the indicator function returning 1 if the predicate a is true and 0 otherwise. The count c_s simply measures the number of times we have seen string s, i.e. $c_s = \sum_{t=1}^{T} I[s_t = s]$. We can think of our predictor as a sequential version of the Parzen windows classifier, which is traditionally used in conjunction with a feature map ϕ and kernel function $k(x, x') = \langle \phi(x), \phi(x') \rangle$. The Parzen windows prediction function g_{pw} is given by

$$g_{\text{pw}}(x) = \text{sign}\left(\sum_{t=1}^{T} y_t k(x_t, x) \right), \tag{7}$$

which takes into consideration each training example. As already mentioned, this is often referred to as the Watson-Nadaraya estimator, which is an estimate of the conditional probability of a class. Note that we no longer have to maintain previous examples as they can be captured by the primal representation of the predictor and we only have to maintain $|\Sigma|^K$ individual weights. In our experiments this remains a feasible primal representation as the maximum alphabet length and string length are set to $|\Sigma| = 6$ and $K = 5$ meaning that $|\Sigma|^K \leq 7776$. In Algorithm 1 we present the sequential BoW algorithm and have introduced two additional variables, a threshold τ and minimum observation number ν. Threshold τ can be interpreted as the excess in probability of a given class occurring that is required to invoke a trading decision. The minimum observation number ν is used to ensure that we can have gathered enough information in order to make a decision. Together these variables control the level of confidence that we have in our trading decision. The dimension of our primal weight vector $\mathbf{w}$ depends on the size of the alphabet Σ and the number of past returns K that we examine, i.e. $|\mathbf{w}| = |\Sigma|^K$. At each time step t we only update a single entry of $\mathbf{w}$, the one corresponding to the particular string observation s_t at that time i.e. w_{s_t}. We construct and update the counts $\mathbf{c}$ in a similar manner.

Algorithm 1 Simple strings (BoW) algorithm

```
$S - BoW(S, R, \Sigma, K, \tau, \nu)$
Require: $S$-string representation, $R$ returns, $\Sigma$ alphabet, $K$ time steps considered, $\tau$ trade threshold, $\nu$ minimum observation number
Initialise weights $w_s = 0$ and counts $c_s = 0$ for each $s \in \Sigma^K$, Profit $R = 0$
for $t = 1 : T$ do
    Observe $s_t$, $c_{s_t}$ compute $f(s_t) = \langle \mathbf{w}, \phi(s_t) \rangle = w_{s_t}$ and $\delta = w_{s_t}/c_{s_t}$
    if $\delta > \tau \cap c_{s_t} > \nu$ then
        Take long position (i.e. buy), $p = 1$
    else if $\delta < -\tau \cap c_{s_t} > \nu$ then
        Take short position (i.e. sell), $p = -1$
    else
        Do not trade, $p = 0$
    end if
    Observe return $r_t$, $y_t = sign(r_t)$
    Update profit $R \leftarrow R + pr_t$
    Update observation counts $c_{s_t} \leftarrow c_{s_t} + 1$
    Update weight vectors $w_{s_t} \leftarrow y_t + w_{s_t}$
end for
```

6.2 String subsequence strategies

In this section we examine an extension of the simple string strategy by measuring the impact of *n*-grams, with and without time-decay, on the probability of class membership. We begin by outlining a sequential NG algorithm (S-NG) that utilises equally-weighted *n*-grams and proceed with introduction of a sequential TDNG algorithm (S-TDNG) that employs the time-decay kernel explained in Section 3.3.

6.2.1 Sequential n-grams strategy: equally-weighted (S-NG)

We could take a step further in the analysis of market alphabet and investigate the subsequences within strings by employing the concept of *n*-grams. Recall that one of the challenges associated with *n*-grams is the choice of the value of *n* and related to it the problem of high dimensionality for high values of *n*. In terms of our strings subsequence strategy, note that the size of the feature space is now $|\Sigma|^n$, where $n \leq K$ is the length of the subsequences that we examine. However our feature space is no longer a binary vector with one non-zero entry, instead there is a non-zero entry for each subsequence that is present in the string. The approach used in Algorithm 1 can be simply adjusted to account for the use of *n*-grams. The observation $s = (s_1^n \dots s_k^n)$ is now decomposed to $k = \max(K - n + 1, 0)$ *n*-grams of length *n*, which results in $f(s) = \frac{1}{k} \sum_{i=1}^{k} w_{s_i^n}$. The weight vector and count updates occur in a synonymous manner to before, where we take each subsequence s_i^n in turn, updating its count $c_{s_i^n}$ and weight $w_{s_i^n}$. To keep our expressions as clear as possible we now drop the superscripts on s_i^n, when the context is clear that we use a fixed length *n*-gram. The confidence of the trading decisions are controlled by the total count $c = \frac{1}{k} \sum_i c_{s_i}$ and the value $\delta = f(s)/c$. The value δ can once again be interpreted as an conditional probability estimate for a given string representation.

6.2.2 Sequential n-grams strategy: time-decay (S-TDNG)

The n-gram feature space that we have described thus far corresponds to the traditional one used in machine learning literature, however this has not been designed to take into consideration any temporal influences that may exist. For example, we would expect that more recent subsequences will have a stronger influence on the likely outcomes and should therefore have a greater weight placed upon them. In the financial literature this empirical phenomenon is often described as the conditional nature of FTS and is assumed across various asset classes and financial markets. To factor this into our representation we can introduce a simple decay function that weights subsequences according to their position in the string,

$$f(s) = \sum_{i=1}^{k} q_i w_{s_i} \quad \text{where} \quad \sum_{i=1}^{k} q_i = 1 \quad \text{and} \quad q_1 \geq q_2 \geq \cdots \geq q_k. \tag{8}$$

Note that in traditional n-grams we effectively have $q_i = \frac{1}{k}$ for each k. We have to make a slight adjustment to the updates, which are now given by $c_{s_i} \leftarrow q_i + c_{s_i}$ and $w_{s_i} \leftarrow q_i y + w_{s_i}$, where $\mathbf{w} = (w_s)_{s \in \Sigma^k}$ maintains a weighted sum of the directional movements associated with each subsequence. We now show that the prediction rule g is equivalent to that of a Parzen windows classifier constructed using this new time decay feature mapping.

Proposition 1 *Let $\phi(s)$ be the feature mapping associated with the time-decay n-gram kernel of length n given by*

$$k(s, s') = \sum_{u \in \Sigma^k} \sum_{i=1}^{k} \sum_{j=1}^{k} q_i q_j I[s_i = u] I[s_j = u], \tag{9}$$

where $\sum_{i=1}^{k} q_i = 1$ and $q_i \geq q_j$ if $i \geq j$. At time T the value of each component of the primal weight vector $(w_{s_i})_{s_i \in \Sigma^k}$ is given by $w_{s_i} = \sum_{t=1}^{T} y_t \sum_{i=1}^{k} q_i I[s_{t,i} = s_i]$, where $s_{t,i}$ corresponds to the i-th n-gram at time t. At time T the prediction function $g(s) = sign(\langle \mathbf{w}, \phi(s) \rangle) = sign\left(\sum_{i=1}^{k} q_i w_{s_i}\right)$ is equivalent to the Parzen windows classifier given by $g_{pw}(s) = sign\left(\sum_{t=1}^{T} y_t k(s, s_t)\right)$.

Proof By expressing the sum over kernel functions in terms of the subsequences present in string s we have

$$\sum_{t=1}^{T} y_t k(s, s_t) = \sum_{t=1}^{T} y_t \sum_{i=1}^{k} q_i \sum_{j=1}^{k} q_j I[s_{t,j} = s_i] = \sum_{i=1}^{k} q_i \sum_{t=1}^{T} y_t \sum_{j=1}^{k} q_j I[s_{t,i} = s_i], \tag{10}$$

which is equivalent to $\sum_{i=1}^{k} q_i w_{s_i}$. Therefore we see that both the Parzen windows classifier and our simple average strategy will return the same prediction. □

Through this proposition we see that we are able to represent the solution learned by the Parzen windows in its primal form using only $|\Sigma|^k$ variables. This allows us to avoid storing past observations in memory and we can efficiently train over as many training samples as we feel is necessary.

Table 1 Descriptive statistics for four currencies as measured on the training set

FX	Mean*	Std.	Skew.	Ex. kurtosis	KS(r)	LB(r)
AUD	−0.090	0.197	−0.445	37.093	0.0000	0.0000
CHF	−0.030	0.133	0.066	12.002	0.0000	0.0022
EUR	−0.017	0.123	−0.026	13.228	0.0000	0.0001
GBP	−0.110	0.128	−0.403	19.915	0.0000	0.0000

Also included are p-values for Kolmogorov-Smirnov normality test as well as for Ljung-Box autocorrelation test (24 lags)

*Value adjustment: 1.0e-2

7 Experimental design

We evaluate the performance of our proposed predictors by using hourly data of four of the most actively traded currency pairs; AUD/USD, CHF/USD, EUR/USD and GBP/USD.[4] The extensive dataset S consists of $T \approx 50,000$ (i.e. $50k$) observations for each currency, covering dates ranging from February 2005 to January 2013. Digressing briefly, the majority of previous experiments using machine learning for FTS prediction have focused on predicting stock returns and often report abnormal returns. We have chosen to focus on currencies due to their permanence and relatively stable prices, rather than the survivorship bias and tendency for an upward drift in prices that exists with stocks taken from indices such as S&P500 or the FTSE100. We investigate the performance from two perspectives: (a) the type of data used for decision making and (b) the complexity of the learning algorithm. The experiments are designed by first splitting full sample into training and test sets $S_{Tr}, S_{Te} \subset S$. Parameter analysis is conducted solely on the training set S_{Tr} by investigating the effect of parameter choice in terms of accuracy scores. After analysing parameter behaviour on S_{Tr} and obtaining optimal specifications for each of the algorithms we then compare their trading performance on the remainder of the data, i.e. test set S_{Te} comprised of subsequent $T \approx 25k$ observations. Test performance is estimated using a number of trading statistics, namely the: (a) cumulative return (*CR*), (b) annualised volatility (*V*), (c) annualised Sharpe ratio (*SR*), (d) gain-to-loss ratio (*GL*), (e) activity ratio (*ACR*), (f) profit per trade (*PPT*), (g) maximum drawdown (*MD*) and (h) maximum drawdown duration (*MDD*) (see Appendix for details).

7.1 Descriptive statistics

Examination of statistical properties of currency pairs is performed on the training set S_{Tr} and indicates the absence of normal distribution and presence of serial autocorrelation in currency returns (Table 1).

Kolmogorov-Smirnov (KS) normality test rejects the null hypothesis H_o of FX returns being normally distributed for all four FX pairs (two-sided test at $\alpha = 1$ % confidence

[4] Since all currencies are USD based we will simply omit the notation for USD and denote the fours currencies as AUD, CHF, EUR and GBP.

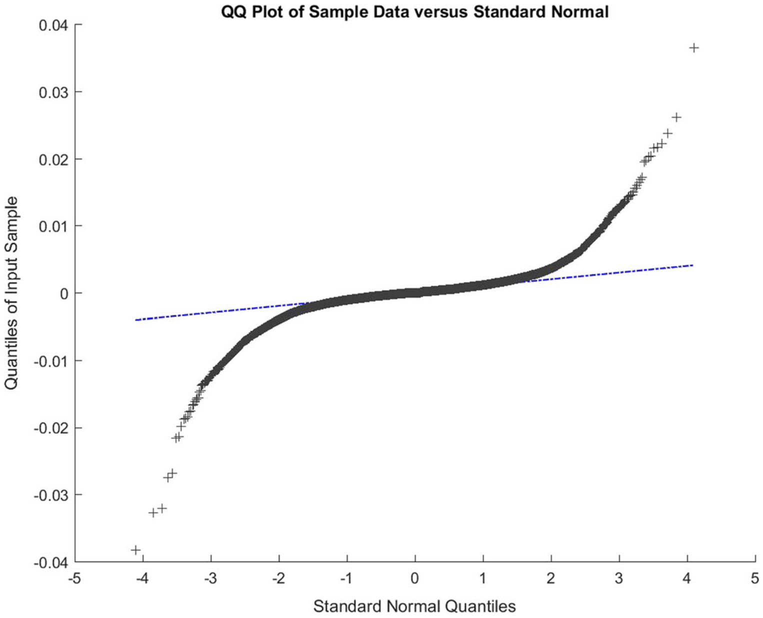


Fig. 2 QQ plot for AUD shows deviation from a standard normal distribution. Excess kurtosis measured on S_{Tr} for AUD is $\kappa = 37.093$

level). The empirical deviation from normality common to all four exchange rates is mirrored in excess kurtosis, i.e. heavy tails of their empirical distributions. Next, Ljung-Box (LB) autocorrelation test rejects null hypothesis H_o of no autocorrelation in the time series indicating that we can not rule out autocorrelation with significant confidence. These findings hold for all four currency pairs, nonetheless deviations from normality and presence of potential autocorrelation are most convincing for GBP and especially for AUD. In fact note that for CHF and EUR the Ljung-Box test rejects the null hypothesis due to inclusion of longer lags but fails to do so for more recent lags. Deviation from normality for AUD is clearly visible in a QQ plot (Fig. 2) that compares empirical sample quantiles and theoretical quantiles from a standard normal distribution (denoted with a straight line). These findings suggest that it may be sensible to take a nonparametric approach to forecasting of intra-day exchange rates. Figure 3 shows presence of potentially significant autocorrelation[5] in AUD returns at $\alpha = 5\ \%$ confidence level, particularly for some of the most recent observations. QQ and ACF plots for remaining three currency pairs were moved to Appendix to prevent cluttering.

8 Training phase: parameter analysis

We begin by investigating parameter dynamics of the algorithms on the training set S_{Tr} comprised of the initial $T \approx 25k$ hourly observations. The remaining $25k$ observations are kept for testing purposes only, i.e. we treat them as theoretically unseen

[5] Or more precisely, a highly significant rejection of the null hypothesis of no autocorrelation.

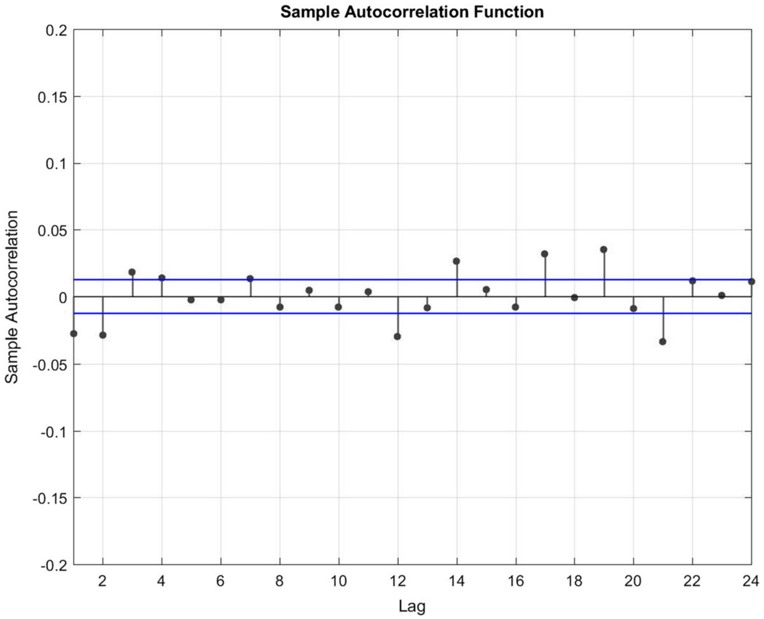


Fig. 3 ACF for AUD hourly returns with straight lines denoting 5 % confidence level (24 lags)

data. This is so as to examine generalisation properties of the algorithms identified as optimal in our training set. Parameters investigated for ν-SVM and FDA are $\nu_i^{svm} = [0.0001,\ 0.001,\ 0.01,\ 0.05,\ 0.1,\ 0.2,\ 0.5]$ and $\lambda_j = 10^{[-3, -2, -1,\ 0,\ 1,\ 2,\ 3]}$, respectively.

8.1 Batch predictors

This section investigates the effects of time series discretisation on predictors' performance. Namely, we train the three batch algorithms, PW, FDA and SVM, using a *sliding windows* approach for an array of alphabet and word lengths and observe the difference in performance to that of classifiers when learning is based on continuous features. The latter are obtained by lagging the original variable where lags correspond to word lengths $K = [1 : 5]$. Sliding windows entails training a classifier on a window of $W_{Tr} = 1500$ observations and then validating it on subsequent window of $W_{Val} = 500$ observations. The reason for adopting this approach is that while a multitude of different parametrisation exist for each of the algorithms, clearly, for such high-dimensional time series data the standard cross-validation approach is not applicable. Hence we train-validate each classifier for a number of parameter values[6] and assess their performance in terms of accuracy on the training set S_{Tr}. Let us use $\boldsymbol{\theta}$ to denote an n-dimensional parameter vector for an individual algorithm, where $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n) \in \Theta \subseteq \mathbb{R}^n$. Then the optimal parameter vector $\boldsymbol{\theta}^*$ is the one that achieves the highest performance $\mathcal{P}$ on the training set, i.e.

$$\boldsymbol{\theta}^* : \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ \mathcal{P} \tag{11}$$

[6] Parameters controlling the degree of regularisation λ and ν in FDA and SVM, respectively.

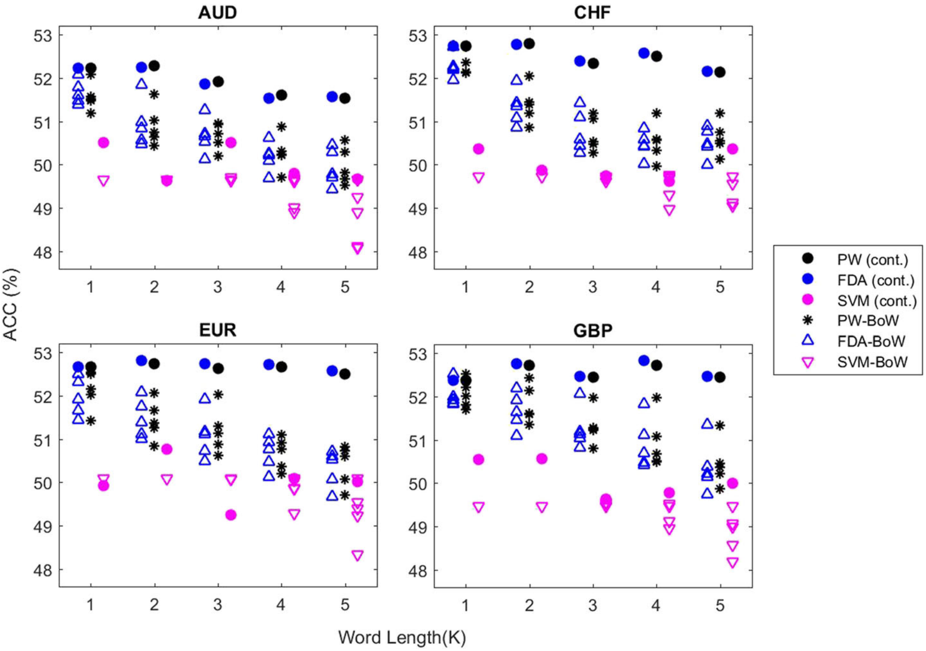


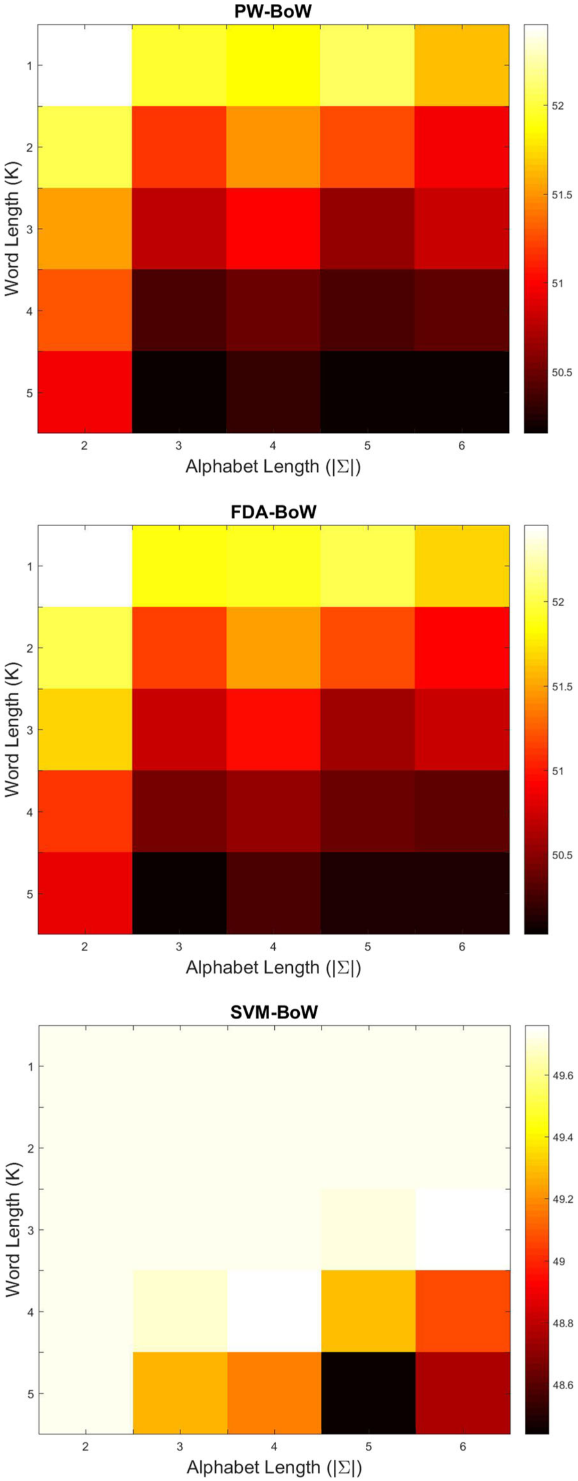
Fig. 4 Performance comparison for PW, FDA and SVM classifiers denoted with *black*, *blue* and *purple markers*, respectively. Algorithms utilise continuous features (for lags analogous to word lengths, i.e. $K = [1 : 5]$) and BoW kernel. BoW are calculated for word lengths $K = [1 : 5]$ and alphabet lengths $|\Sigma| = [2 : 6]$

where at this stage we estimate performance in terms of accuracy $\mathcal{P} := Acc$. Parameter specification $\boldsymbol{\theta}^*$ that achieved the highest training accuracy is ultimately tested on the theoretically unseen set S_{Te}.

8.1.1 Features: continuous vs. discrete

Our initial experiments entailed analysis of the discretisation impact on performance of batch classifiers. Figure 4 shows accuracy achieved by PW, FDA and SVM when continuous features were used as inputs vs. when a simple BoW kernel was used as a similarity measure between the strings. Word and alphabet lengths applied were $K = [1 : 5]$ and $|\Sigma| = [2 : 6]$, respectively. The former are plotted along the x-axis with different markers at each word length representing different alphabet lengths. Note that for FDA and SVM, both of which were trained for a range of respective regularisation parameters, these figures represent their performance averaged across all parameter values for simplicity of presentation. Clearly, the effects of FTS discretisation are counter-productive as we observe performance deterioration across both algorithms as well as word/alphabet lengths. SVM noticeably achieves lower accuracies across the board for both continuous and string features. Also, it seems that extending word lengths further decreases information content as performance deteriorates for longer strings. Figure 5 displays accuracies across a combination of word/alphabet lengths for each of the three algorithms using string (BoW) features averaged across all four FX pairs. Clearly, shorter word and alphabet combinations are preferred to longer ones as accuracy deteriorates for the latter case.

Fig. 5 Word vs. alphabet length: accuracies (in %) averaged across all four FX pairs for PW-BoW (*top*), FDA-BoW (*middle*) and SVM-BoW (*bottom*)



8.1.2 *Effect of time-decay*

Next we investigate the performance of string algorithms when a more detailed similarity kernels are used, i.e. n-grams (NG) and time-decay n-grams (TDNG). Moreover, we are interested in the effect of applying a decay parameter $d = \tilde{q}_1/\tilde{q}_k$ that effectively determines the degree of importance between the first and last observation ($d = 1$ for equally-weighted n-grams). We keep the length of subsequence fixed at $n = 1$. Figure 6 shows that accounting for time-decay (here fixed at $d = 5$) is in fact beneficial as algorithms on average record higher accuracies with TDNG compared to NG kernel (red markers). Understandably, there is no difference in performance for $K = 1$ word length as time-decay does not come into effect for a single lag (which results in TDNG markers overwriting red NG markers). Nonetheless, as shown in Fig. 7 even more complex string kernels do not manage to consistently outperform algorithms trained on continuous features despite showing encouraging results.

8.2 Sequential predictors

This section focuses on parameter analysis of sequential string algorithms on the training sample S_{Tr}. We begin by comparing the effects of word, alphabet and n-gram lengths on performance. We continue by investigating the dynamics of decay, threshold and minimum observation parameters.

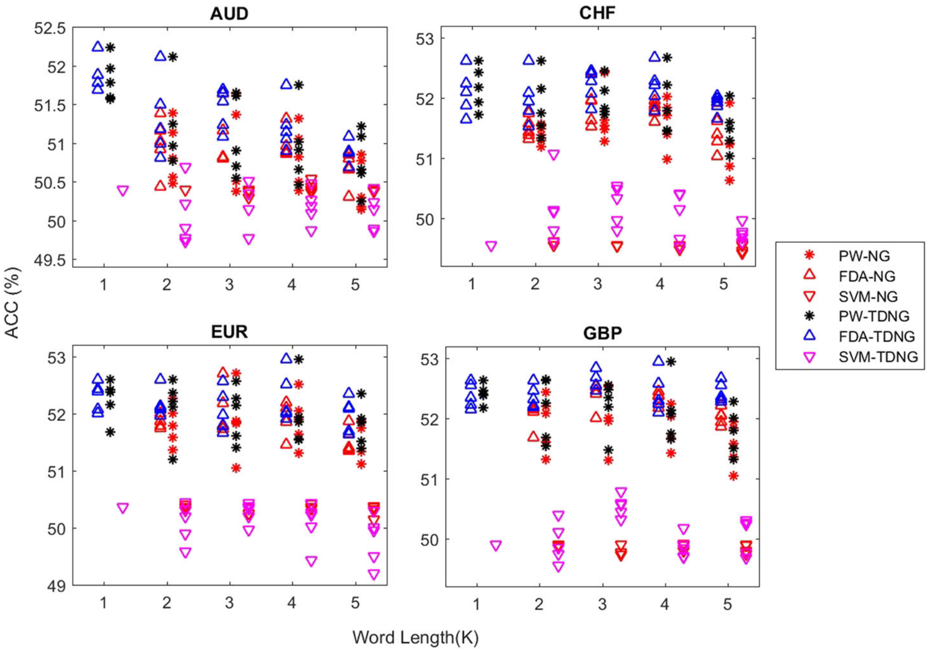


Fig. 6 Performance of PW, SVM and FDA classifiers when equally-weighted NG kernel is applied (*red markers*) compared to TDNG kernel with decay (here fixed at $d = 5$). String parameters are $|\Sigma| = [2 : 6]$, $K = [1 : 5]$ and $n = 1$

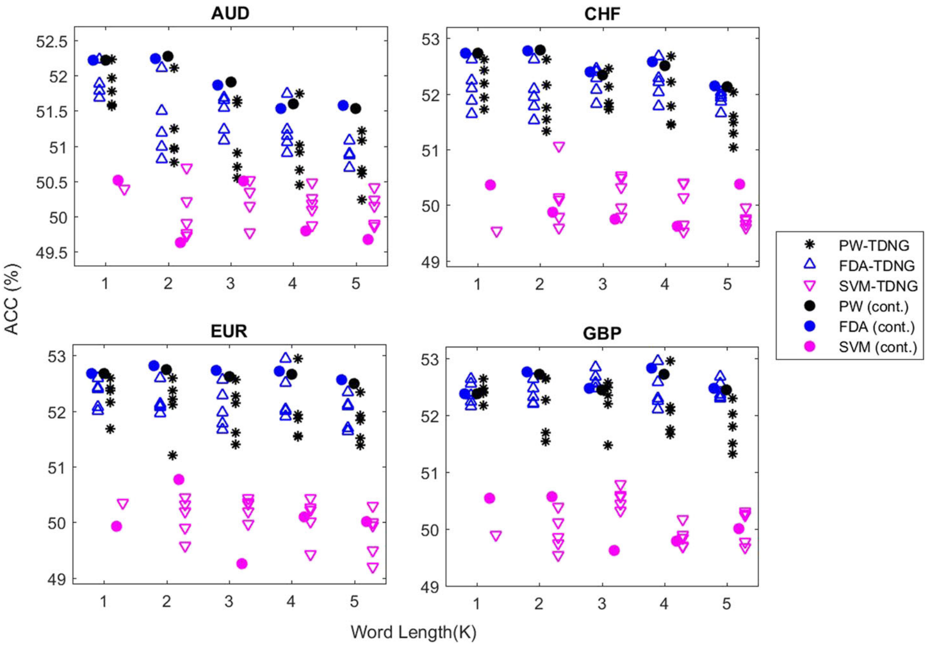


Fig. 7 Performance of PW, SVM and FDA classifiers when trained on continuous features compared to the same batch algorithms utilising a TDNG kernel with decay (here fixed at $d = 5$). String parameters are $|\Sigma| = [2 : 6]$, $K = [1 : 5]$ and $n = 1$

8.2.1 String length analysis: words vs. alphabet vs. n-grams

Figure 8 displays training performance for S-BoW algorithm for $\Sigma = [2 : 6]$ and $K = [1 : 5]$ combinations. These results seem to reinforce the preference for shorter word and alphabet combinations. Furthermore, we run S-BoW for two cases, namely when we do not impose advice of confidence parameters, i.e. τ and ν are equal to zero (blue markers), and when we do impose confidence parameters in this case $\tau = 0.05$ and $\nu = 50$ (red markers; algorithms followed by a suffix ‘A’). The effect of using advice is clearly positive and seemingly significant. Moreover, not only do S-BoW-A outperform their counterparts that do not utilise advice, they also clearly outperform the batch PW strategy that learned using continuous data (black circles), a result that none of the three batch algorithms managed to achieve regardless of the string kernel type. We continue by comparing S-NG and S-TDNG strategies (both without advice) for the same word and alphabet combinations whilst adding the subsequence dimension, i.e. $n = [1, 2, 3]$. As depicted in Fig. 9, the S-NG accuracies denoted with red markers are visibly lower compared to the results achieved by the S-TDNG strategy that utilises the time-decay n-gram kernel ($d = 5$). Moreover, this result holds across the word, alphabet and n-gram combinations. Lastly, we examine the effects of confidence parameters τ and ν for the case of S-TDNG ($d = 5$) and plot its performance along the batch PW algorithm learned on continuous data (Fig. 10). We observe that when no advice is imposed (red markers) the S-TDNG strategy manages to match the performance of a batch PW algorithm. However, as was the case with S-BoW-A in the beginning of this section, imposing advice improves performance of the algorithm across the word, alphabet

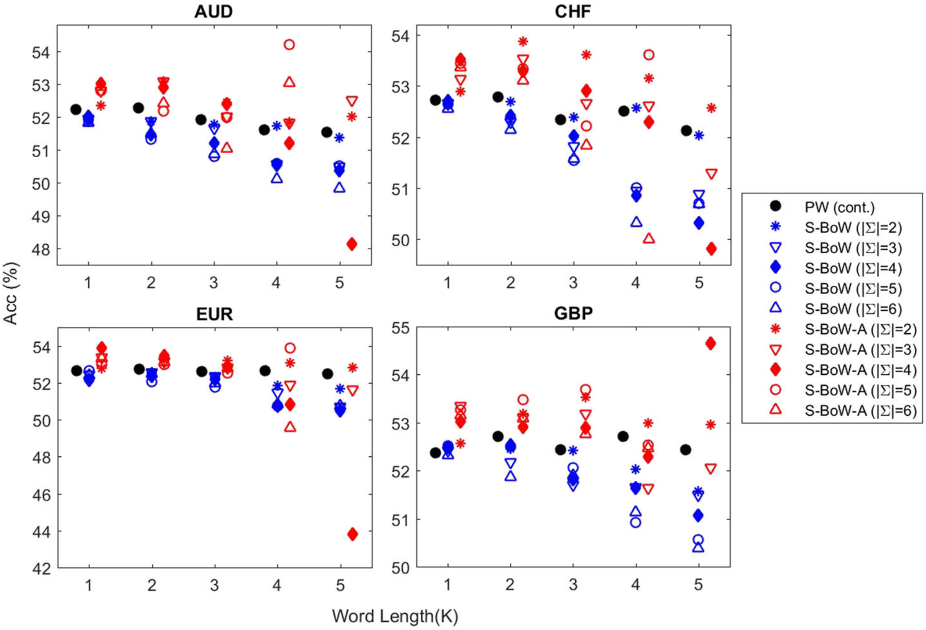


Fig. 8 Sequential S-BoW with advice (*red markers*) and without advice (*blue markers*). Accuracies across various word and alphabet lengths. Batch PW with continuous data is added for comparison (*black circles*)

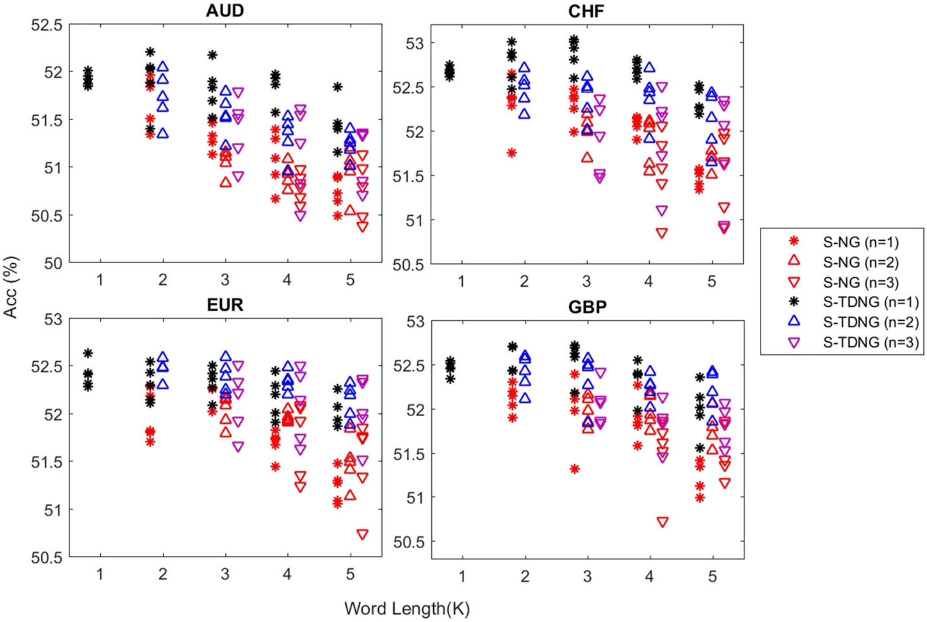


Fig. 9 Sequential S-NG equally-weighted (*red markers*; $d = 1$) and S-TDNG with time-decay (*black*, *blue* and *purple markers*). The decay is fixed at $d = 5$. Accuracies across various word, alphabet and *n*-gram lengths

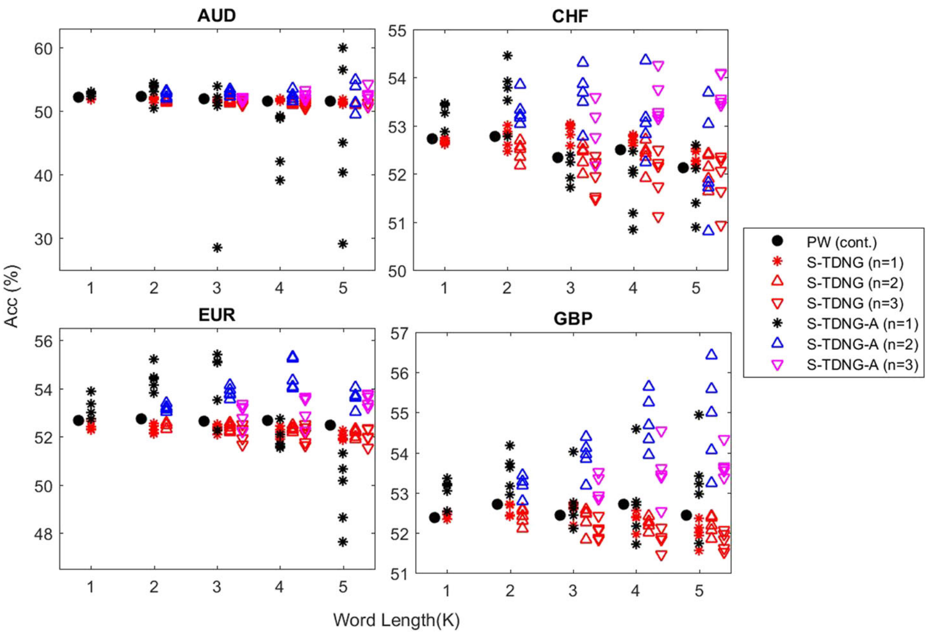


Fig. 10 Time-decay S-TDNG without advice (*red markers*) and S-TDNG with advice (*black*, *blue* and *purple markers*). The decay is fixed at $d = 5$. Accuracies across word, alphabet and n-gram lengths. Batch PW with continuous data for comparison (*black circles*)

and n-gram combinations. The effect of confidence parameters as well as the decay parameter on the classification performance is clearly important and so in the following sections we examine the effects of τ, ν and d in more detail.

8.2.2 Time decay n-grams: decay analysis

We investigate the influence of decay parameter in time-decay kernel on the performance of the sequential string algorithms. We experimented using decay factors $d = [1, 5, 10, 15, 25, 35, 50, 100]$ for two types of parameter combinations, i.e. of high (H) and low dimensionality (L). High-dimensional set had alphabet and word lengths fixed at $|\Sigma| = 5$ and $K = 4$ with n-gram lengths equal to $n = [1, 2, 3]$. Low-dimensional set on the other hand consisted of parameter values $|\Sigma| = 4$, $K = 3$ and $n = [1, 2]$. Threshold τ and minimum observation ν parameters are both fixed at zero. To compute the weightings q_i used with the function $f(s) = \sum_i q_i w_i$ we begin by letting $\tilde{q}_1 = 1$ and recursively compute $\tilde{q}_{i+1} = \tilde{q}_i c$. The decay factor d states that $\tilde{q}_1/\tilde{q}_k = d$, where k is the number of n-grams present in the string. Therefore using the recursions we observe that $\tilde{q}_1 = d\tilde{q}_1 c^{k-1}$ and that $c = e^{-\frac{\log d}{k-1}}$. A final normalisation $q_i = \tilde{q}_i / \sum_j \tilde{q}_j$ ensures the weights q_i sum to one. The results of the experiments quite clearly show a preference for larger decay factors as accuracies increase as we move from equally-weighted S-NG predictor ($d = 1$) towards higher decays along the x-axis (Fig. 11). Also, lower parameter dimensions (star markers) seem to be preferable as well as less sensitive to the choice of d. Note, however that after a certain point the decay value becomes less relevant and in some cases even counter-productive.

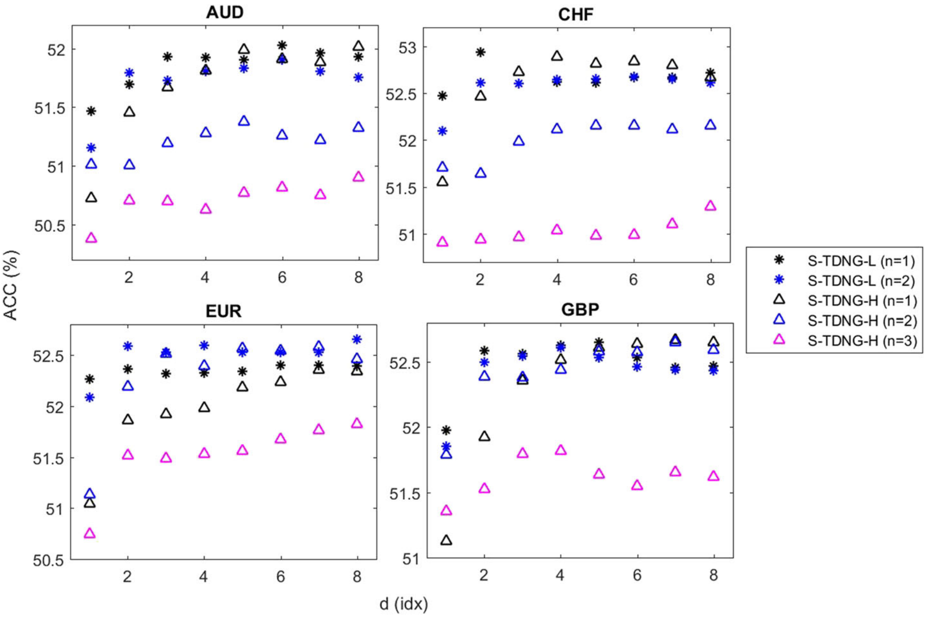


Fig. 11 Influence of time decay on performance. Both minimum observation ν and threshold τ parameters are set to zero. Reported are accuracies for different model combinations with n-grams of lengths $n = [1, 2, 3]$ (index related to each individual decay value displayed on x-axis)

8.2.3 *Threshold analysis* (τ)

The role of threshold τ is to increase forecast guarantees of our string strategies. Experiments were performed by fixing the remaining parameters and observing classification accuracies for different threshold values $\tau = [0 : 0.01 : 0.2]$. As was the case in previous Section 8.2.2 we investigate performance for high (H) and low (L) dimensional features. The comparison between all three string subsequence strategies, S-BoW, S-NG and S-TDNG, is shown in Figure 12, with decay parameter in time-decay n-gram strategy held fixed at $d = q_1/q_K = 5$ and minimum observations held at $\nu = 0$. We observe slightly higher accuracies for S-TDNG algorithms, however note that after a certain τ value performance becomes increasingly volatile. Also, as was the case in previous section, lower dimensionalities are preferred to higher ones regardless of the strategy type.

8.2.4 *Confidence analysis* (ν)

Here, too, we gradually increase the minimum number of observations determined by $\nu = [0 : 50 : 1000]$. Similarly to τ, parameter ν governs confidence of our forecasts. Remaining parameters are held constant at $|\Sigma| = 4$, $K = 3$ and $n = 3$ with threshold fixed at $\tau = 0$ and decay ratio in time-decay n-grams fixed at $q_1/q_K = 5$. Time-decay n-grams strategy S-TDNG (purple markers) achieves higher accuracy scores compared to both S-BoW (black markers) as well as S-ND (blue markers) in three out of four cases (Fig. 13). Note, however that we report results for low-dimensional features only as for

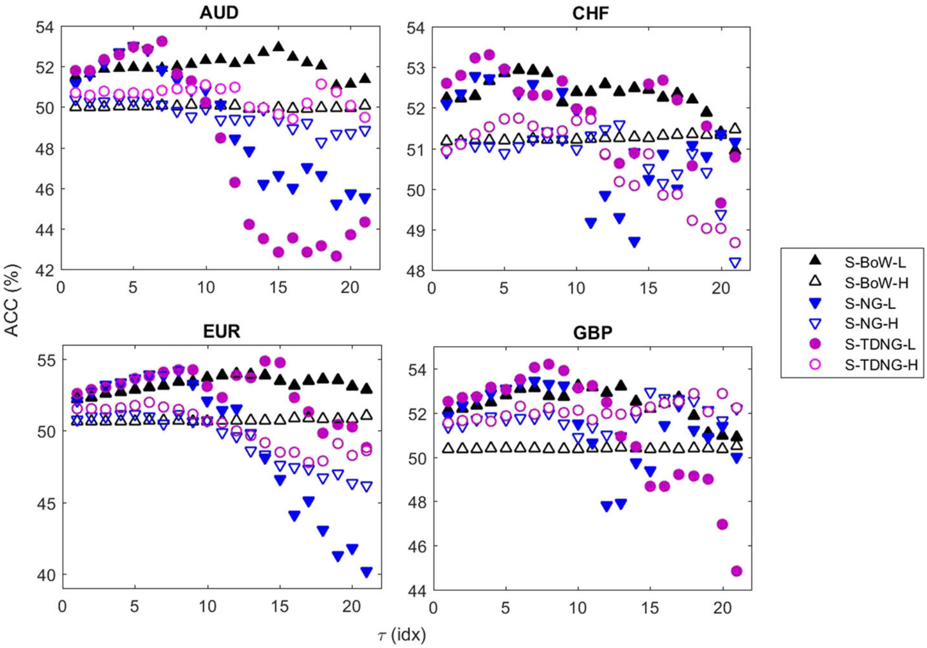


Fig. 12 Threshold analysis: accuracies achieved across a range of τ values (x-axis denoting their respective indices). Algorithms S-BoW (*black markers*), S-NG (*blue markers*) and S-TDNG (*purple markers*) seem to favour lower dimensional feature spaces (*full markers*) over higher ones (*empty markers*)

high-dimensional cases algorithms would not always be able to satisfy the conditions. S-BoW in particular showed high sensitivity to higher values of ν (even for low dimensional features as shown in Fig. 13), a direct result of its inability to find a high-enough number of matches so as to cross the required ν value. Our detailed parameter analysis indicates that while sequential string algorithms do manage to achieve superior performance on the training set, they also exhibit sensitivity to parameters when these are increased to extremes. Hence we restrict our optimisation procedures to lower parameter values where our strategies exhibit stable training performances.

9 Testing phase: generalisation properties

Finally, we test all optimal classifiers, batch and sequential, on the test set S_{Te}, that is the second set of $T = 25k$ hourly observations, so as to investigate their ability to generalise on theoretically unseen data. We also compare their performance to a naive strategy whose directional prediction is based on the last hourly return. In other words, in such a strategy a trading signal for the upcoming trading session is determined solely by the direction of the last available observation, i.e. $s_t = \text{sign}(y_{t-1}) \in \{-1, 1\}$ for a naive momentum strategy and $s_t = -\text{sign}(y_{t-1}) \in \{-1, 1\}$ for a naive contrarian strategy. We examine these results together with the maximum accuracy results achieved by the optimal algorithms on the training set S_{Tr}. Performance on the test set S_{Te} is again assessed via accuracy, however,

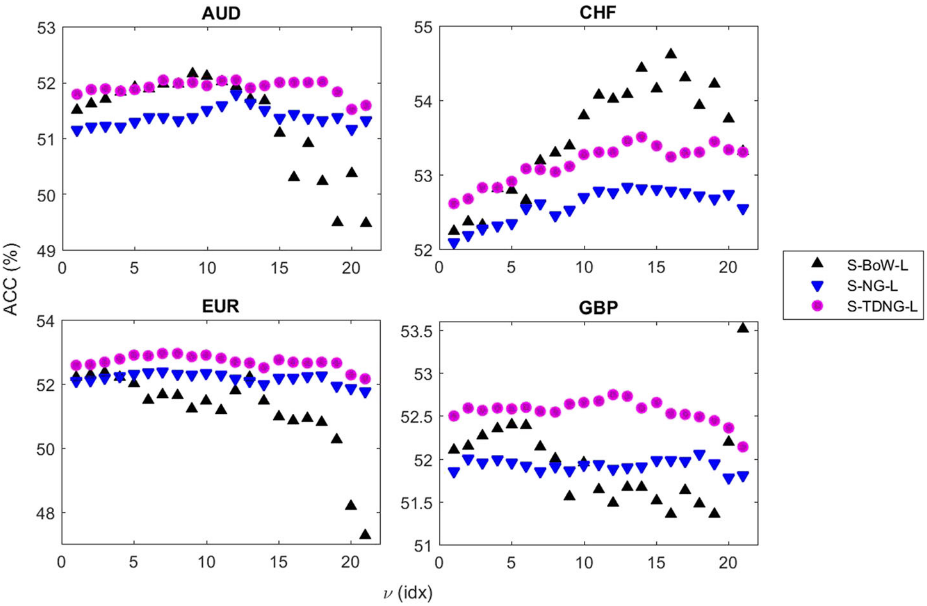


Fig. 13 Minimum observation analysis: S-BoW (*dark markers*) vs. S-NG (*blue markers*) and S-TDNG (*purple markers*). Respective indices for ν values are denoted on x-axis

we also employ the following trading metrics: (a) cumulative return (*CR*), (b) annualised volatility (*V*), (c) annualised Sharpe ratio (*SR*), (d) gain-to-loss ratio (*GL*), (e) activity ratio (*ACR*), (f) profit per trade (*PPT*), (g) maximum drawdown (*MD*) and (h) maximum drawdown duration (*MDD*) (see Appendix for details).

9.1 Classification performance

Table 2 conveys maximum training accuracies achieved by each best classifier $f(\boldsymbol{\theta^*})$. We witness a gradual improvement in performance as we move from utilising BoW towards NG and TDNG kernels. Also, while continuous features add information value to the extent that their algorithms perform at par with string-based classifiers, this does not seem to be the case with S-TDNG that achieves the highest accuracy in three out of four cases (AUD, CHF and GBP). Also, we observe that on average most algorithms outperform both naive strategies with an exception of SVMs. Note that in most string kernel cases, PW and FDA classifiers achieve almost identical results. This indicates that optimal FDA algorithms employ the highest regularisation value $\lambda = 10^3$ which reduces FDA to a PW classifier since covariances become practically spherical and FDA's property to maximise class separation by readjusting the coordinate scheme of the data becomes ineffective. Next, Table 3 shows performance of the same *best* algorithms when used for prediction on the unknown data set S_{Te}. We observe a slight deterioration in performance across the algorithms, nonetheless the best performing predictors again seem to be sequential string algorithms (especially S-TDNG) as well as batch PW and FDA predictors.

Table 2 Maximum accuracies achieved by predictors on the training set S_{Tr}

$f(\boldsymbol{\theta}^*)$	ACC (%)			
	AUD	CHF	EUR	GBP
Naive-M	47.85	47.10	47.63	47.54
Naive-C	52.12	52.90	52.37	52.46
PW (cont.)	52.28	52.78	52.74	52.71
FDA (cont.)	52.29	52.86	52.83	52.81
SVM (cont.)	51.40	50.99	51.19	50.89
PW-BoW	52.07	52.72	52.53	52.53
FDA-BoW	52.07	52.72	52.60	52.53
SVM-BoW	50.34	50.60	50.57	50.45
PW-NG	52.23	52.63	52.70	52.63
FDA-NG	52.23	52.63	52.70	52.63
SVM-NG	50.98	51.44	51.10	51.34
PW-TDNG	52.23	52.67	**52.95**	52.95
FDA-TDNG	52.23	52.67	**52.95**	52.95
SVM-TDNG	51.38	51.72	51.37	51.30
S-BoW	52.10	53.14	52.71	52.74
S-NG	52.11	53.16	52.72	52.88
S-TDNG	**52.43**	**53.25**	52.86	**53.15**

Table 3 Accuracies achieved on the test set S_{Te} by *best* training predictors

$f(\boldsymbol{\theta}^*)$	ACC (%)			
	AUD	CHF	EUR	GBP
NaiveM	48.43	46.55	47.11	47.27
NaiveC	51.57	53.45	**52.89**	52.73
PW (cont.)	51.26	53.27	52.48	51.85
FDA (cont.)	51.23	53.39	52.70	51.98
SVM (cont.)	50.04	49.89	49.79	50.19
PW-BoW	51.40	53.52	52.61	**52.80**
FDA-BoW	51.40	53.52	52.50	**52.80**
SVM-BoW	50.33	50.62	50.09	50.20
PW-NG	51.40	53.52	51.58	**52.80**
FDA-NG	51.40	53.52	51.58	**52.80**
SVM-NG	49.61	50.54	50.58	49.98
PW-TDNG	51.40	52.76	52.12	52.54
FDA-TDNG	51.40	52.76	52.12	52.54
SVM-TDNG	49.34	50.56	50.17	50.55
S-BoW	51.11	53.45	52.62	52.11
S-NG	51.34	53.60	52.65	52.58
S-TDNG	**51.58**	**53.68**	52.13	52.66

Table 4 AUD: trading performance on the test set S_{Te}

	AUD								
	CR*	V*	SR	t-Stat	GL	ACR	PPT*	MD*	MDD
Naive-M	−56.13	13.14	−1.14	−2.13	0.94	1.00	−0.25	66.69	911
Naive-C	56.13	13.14	1.14	2.13	1.07	1.00	0.25	15.04	194
PW (cont.)	33.23	12.60	0.75	1.41	1.05	0.94	0.17	18.24	332
FDA (cont.)	29.77	12.60	0.67	1.26	1.05	0.94	0.15	18.19	332
SVM (cont.)	−19.87	12.60	−0.45	−0.84	1.00	0.94	−0.10	33.81	841
PW-BoW	31.93	12.60	0.72	1.35	1.06	0.94	0.16	22.16	353
FDA-BoW	31.93	12.60	0.72	1.35	1.06	0.94	0.16	22.16	353
SVM-BoW	11.31	12.60	0.26	0.48	1.01	0.94	0.06	24.30	547
PW-NG	31.93	12.60	0.72	1.35	1.06	0.94	0.16	22.16	353
FDA-NG	31.93	12.60	0.72	1.35	1.06	0.94	0.16	22.16	353
SVM-NG	−17.75	12.60	−0.40	−0.75	0.98	0.94	−0.09	35.27	322
PW-TDNG	31.93	12.60	0.72	1.35	1.06	0.94	0.16	22.16	353
FDA-TDNG	31.93	12.60	0.72	1.35	1.06	0.94	0.16	22.16	353
SVM-TDNG	−56.61	12.60	−1.28	−2.40	0.97	0.94	−0.28	58.41	882
S-BoW	44.79	13.12	0.91	1.76	1.05	1.00	0.65	15.74	325
S-NG	48.95	13.14	0.99	1.92	1.06	1.00	0.63	17.70	325
S-TDNG-25	62.84	13.14	1.28	2.47	1.07	1.00	0.81	17.43	332

9.2 Trading performance

In this section we assess the quality of the algorithms from a trading perspective. We estimate trading performance using a number of metrics separately for each of the four exchange rates. We pay special attention to the annualised Sharpe ratio which communicates the risk-adjusted returns and similarly to the *t*-statistic (also reported) essentially signals the confidence of the results, i.e. to what extent is a mean return different from zero. Note however, that due to return distributions exhibiting quite high excess kurtosis estimates, the reliability of these statistics should be observed with some forethought. The names of the algorithms in bold are denoting those with the highest *t*-statistic, and hence Sharpe ratio, achieved on that particular sample (exchange rate). Table 4 shows results for AUD with results for the other three currencies moved to Appendix for higher clarity. We observe that in the case of AUD the best performing strategy according to Sharpe ratio as well as other criteria such as cumulative profit (CR) and profit-per-trade PPT is S-TDNG. Also, it seems running a naive contrarian strategy, S-BoW and S-NG resulted in returns significant at 5 % confidence level. Runing a naive momentum strategy on other hand would have had quite negative consequences as would relying on predictions of almost any SVM classifier. The best trading strategies on CHF, EUR and GBP are S-NG, naive contrarian and string-based batch classifiers (PW and FDA) respectively, with naive momentum and SVM based predictors consistently performing worse. Overall, trading performance seems to resemble results when comparing accuracies, a sensible result as our trading does not include position sizing of any kind. Note however, that the flexibility of our approach allows us to easily change our choice of performance criterion $\mathcal{P}$ something to be investigated in the future.

10 Conclusions and future research

This paper investigates the idea of time series discretisation and utilises text or string kernels to learn about the underlying market dynamics. It proposes a new type of *n*-gram kernel that takes into account conditional nature of FTS data, namely the time-decay *n*-grams. We first approach the problem by using batch versions of the Parzen windows, Fisher discriminant analysis and support vector machine algorithms, and analyse the effect of using discrete representation rather than continuous real-valued inputs to predict the direction of hourly foreign exchange returns. Our experiments show that the Parzen windows and Fisher discriminant analysis, both of which focus on centres of the distribution rather than the margins (contrary to SVMs), are effectively outperforming the support vector machine classifier for both continuous and discrete data. We also observed that discretisation does not automatically improve performance, but much depends on the choice of kernel function used to learn the *market alphabet*. Namely, the only kernel able to match and to some extent outperform batch classifiers with continuous inputs, is the time-decay *n*-grams kernel proposed in this paper that captures temporal influences of string subsequences. However, due to batch classification algorithms being to some extent hampered by their ability to only consider a finite number of training samples, we consequently build on these results and introduce a novel approach to FTS forecasting based on a simple string representation and sequential learning with market alphabet representation that resembles incremental Parzen windows algorithm. We examine the design of this trading strategy from two perspectives, namely the complexity of the underlying algorithm and the representation of the underlying time series used in the decision making process. We show that time-decay kernel can be evaluated efficiently using a simple weighted averaging process that is equivalent to the Parzen windows classifier using that kernel. The results of these experiments suggest that a simple string representation coupled with an averaging process is capable of outperforming more exotic approaches, whilst supporting the idea that when it comes to working in high noise environments often the simplest approach is the most effective.

Appendix

Performance metrics

Cumulative return

If T denotes the number of trading instances (i.e. hours) and r_t is the strategy return at time t then we calculate cumulative return as:

$$R = \sum_{t=1}^{T} r_t \tag{12}$$

Volatility

Unconditional volatility of a trading strategy is calculated as a (square root of) sample variance of the strategy's returns:

$$s^2 = \frac{1}{T-1}\sum_{t=1}^{T}(r_t - \bar{r})^2 \tag{13}$$

Sharpe ratio

Sharpe ratio is a risk-adjusted performance metric proposed by Sharpe (1966). In its original form it is equal to:

$$SR = \frac{\mu(r)}{\sigma(r)} \tag{14}$$

Here $\mu(r)$ is an average excess return of the strategy over a benchmark (such as a risk-free rate for example) and $\sigma(r)$ is a standard deviation of excess returns. We assume a risk-free rate of $r_f = 0$ so the average excess return of the strategy equals the average return of the strategy. Note, that we are interested in the annualised estimate of the Sharpe ratio which requires us to scale the estimate with a square root of the number of trading instances in a year. For example for the case of hourly data with 252 business (i.e. trading) days with $24h/day$ the annualised Sharpe ratio (ASR) is equal to $ASR = \sqrt{252 * 24} \times SR$. Note, that the same principle applies when calculating the annualised estimate of volatility.

T-statistic

We use a simple t-statistic to test for the significance of the time series of strategy returns. Namely, we test the hypothesis that the population mean return significantly differs from zero, i.e. $H_o : \mu_o = 0$. This single-sample metric is essentially a Sharpe ratio[7] multiplied by the square root of the sample size T used in the underlying experiment.

$$t = \frac{\mu(r) - \mu_o}{\sigma(r)/\sqrt{T}} = \sqrt{T}\frac{\mu(r)}{\sigma(r)}, \tag{15}$$

where $\mu(r)$ is the sample mean of the strategy return r and $\sigma(r)$ is its sample standard deviation.

Gain-to-loss ratio

We can write the equation for gain-to-loss ratio as

$$GL = \frac{\sum_{t=1}^{T^+} f(r)}{\sum_{t=1}^{T^-} g(r)} \tag{16}$$

[7] Sharpe ratio as expressed in its basic form, i.e. not annualised or adjusted in any way.

where function $f(r)$ is a boolean operator for positive returns

$$f(r) = \begin{cases} 1 & \text{if} \quad r^{+} \\ 0 & \text{if} \quad \text{otherwise} \end{cases} \tag{17}$$

and function $g(r)$ is a boolean for negative returns

$$g(r) = \begin{cases} 1 & \text{if} \quad r^{-} \\ 0 & \text{if} \quad \text{otherwise} \end{cases} \tag{18}$$

and T^{+} and T^{-} denote the number of trading instances with positive (r^{+}) and negative (r^{-}) returns, respectively. Hence, GL ratio effectively measures the number of winning trades relative to the number of losing trades.

Activity ratio

If T denotes the number of trading instances and N denotes the number of all instances in a sample then activity ratio is simply calculated as:

$$ACR = \frac{T}{N} \tag{19}$$

Profit-per-trade

We obtain profit-per-trade by averaging cumulative return of a strategy across the number of active trading instances:

$$PPT = \frac{\sum_t r_t}{T} \tag{20}$$

Maximum drawdown and maximum drawdown duration

Maximum drawdown measures the maximum cumulative return R loss suffered by the strategy in the entire trading period before recuperating losses. We calculate it as Dunis and Williams (2003):

$$MD = \min_{t=1,\ldots,T} \left(R_t - \max_{i=1,\ldots,t} (R_i) \right) \tag{21}$$

Maximum drawdown duration (MDD) is equal to the number of trading instances that the drawdown spans over, i.e. the amount of time that it takes for the strategy to reach the maximum cumulative return of the past.

Descriptive statistics (Figs. 14, 15 and 16)

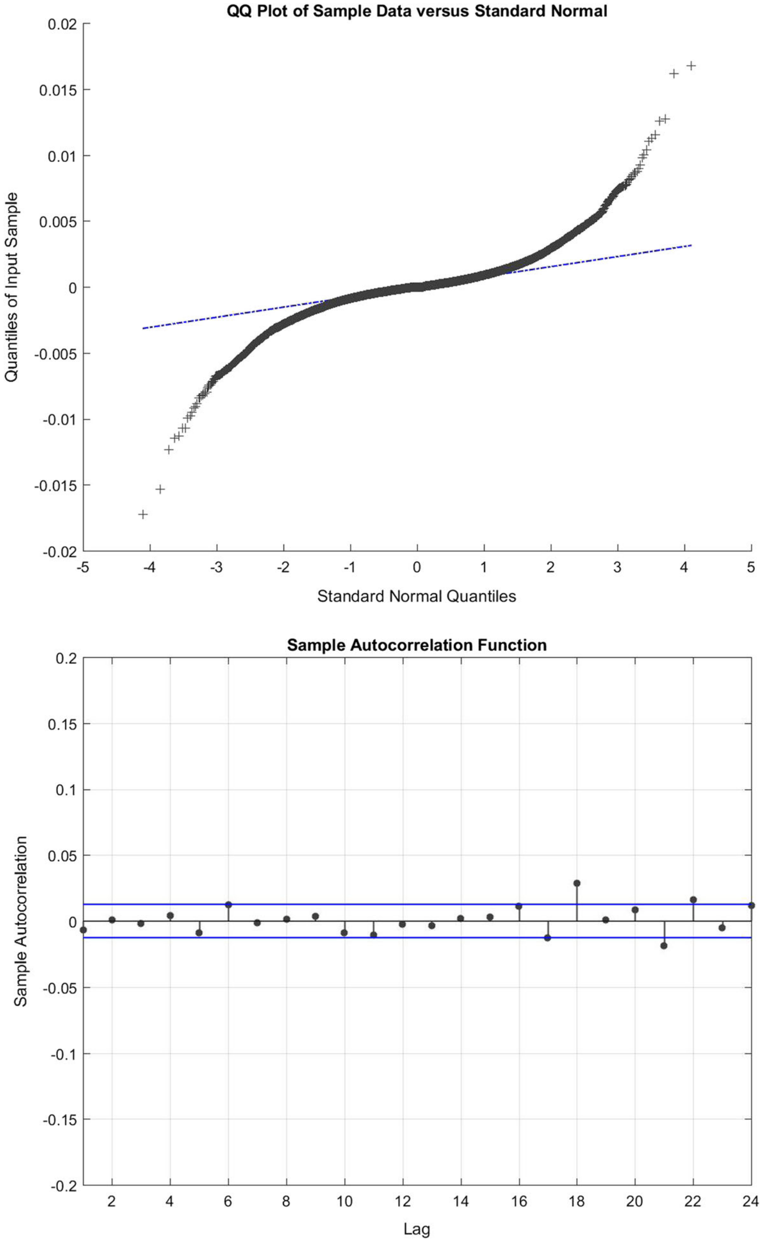
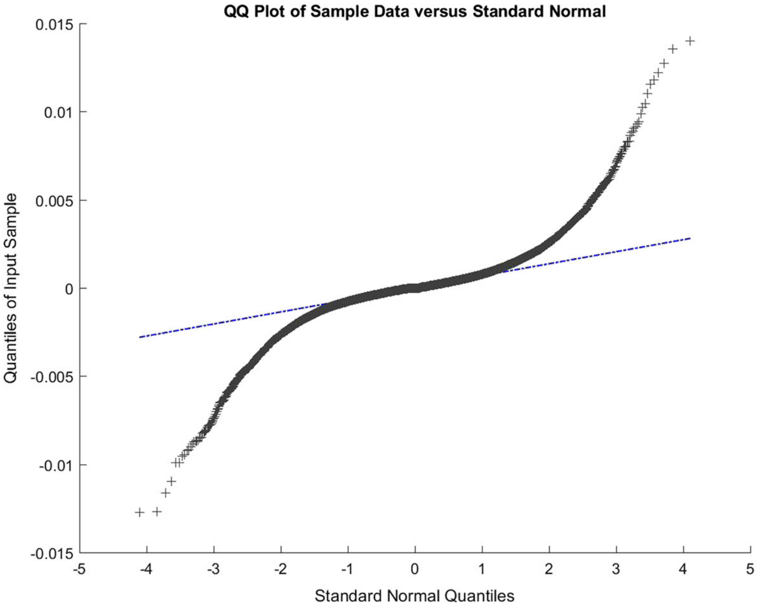


Fig. 14 *Top*: QQ plot for CHF shows deviation from a standard normal distribution. Excess kurtosis measured on S_{Tr} for CHF is $\kappa = 12.002$. *Bottom*: ACF for CHF hourly returns with straight lines denoting 5 % confidence level



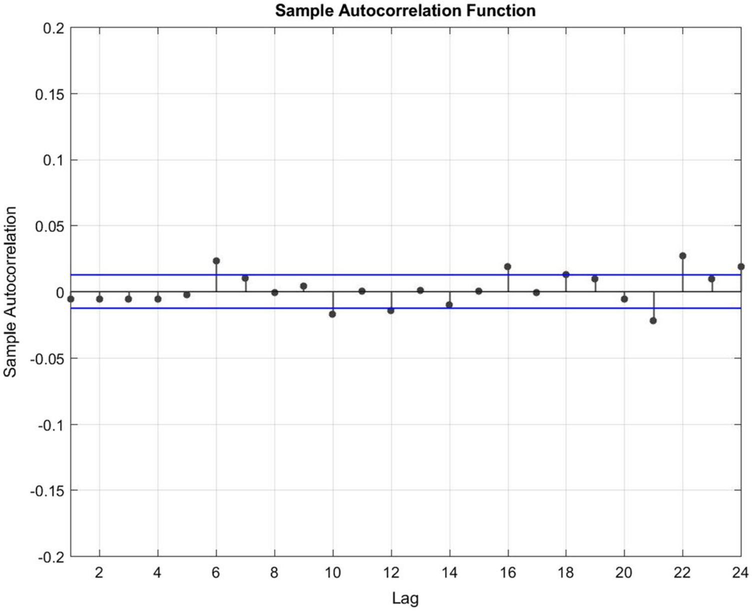
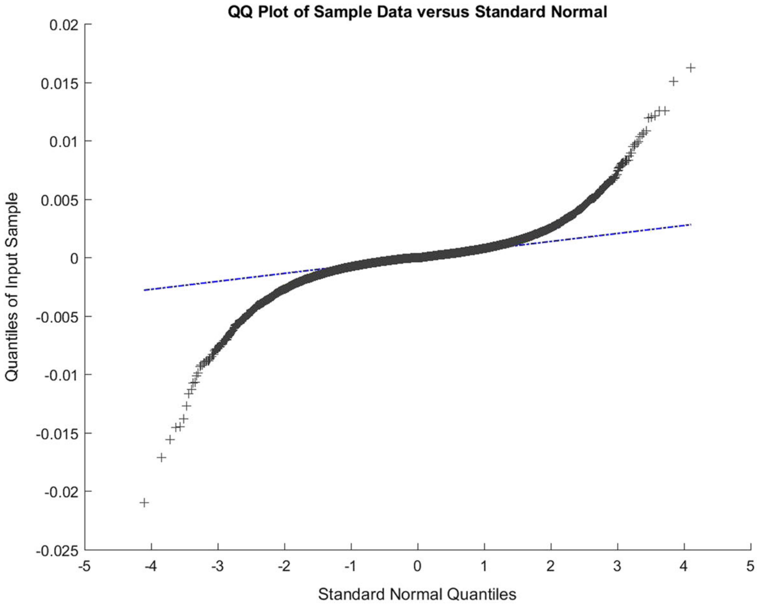


Fig. 15 *Top*: QQ plot for EUR shows deviation from a standard normal distribution. Excess kurtosis measured on S_{Tr} for EUR is $\kappa = 13.228$. *Bottom*: ACF for EUR hourly returns with straight lines denoting 5 % confidence level



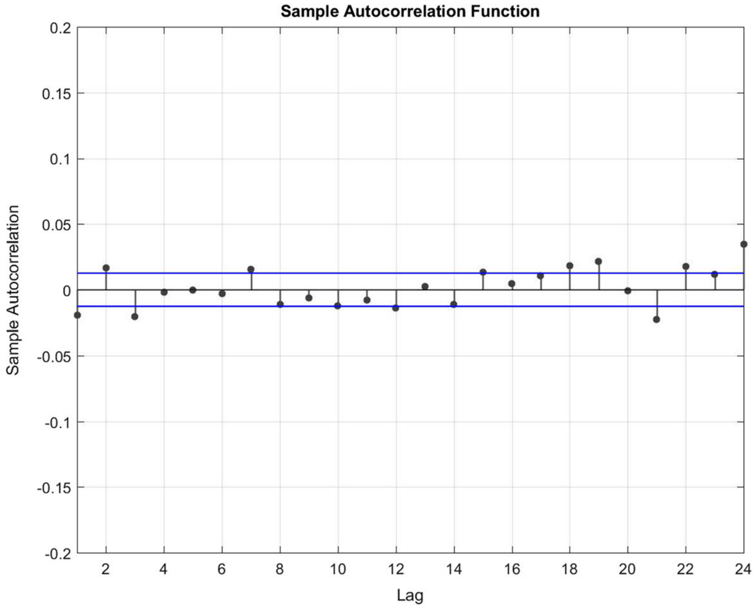


Fig. 16 *Top*: QQ plot for GBP shows deviation from a standard normal distribution. Excess kurtosis measured on S_{Tr} for GBP is $\kappa = 19.915$. *Bottom*: ACF for GBP hourly returns with straight lines denoting 5 % confidence level

Trading performance (Tables 5, 6 and 7)

Table 5 CHF: trading performance on the test set S_{Te}

	CHF								
	CR*	V*	SR	t-Stat	GL	ACR	PPT*	MD*	MDD
Naive-M	−91.21	11.81	−2.05	−3.84	0.87	1.00	−0.40	93.42	948
Naive-C	91.21	11.81	2.05	3.84	1.15	1.00	0.40	11.61	85
PW (cont.)	85.33	11.59	2.09	3.92	1.14	0.94	0.43	10.99	83
FDA (cont.)	81.70	11.59	2.00	3.75	1.15	0.94	0.41	11.24	128
SVM (cont.)	−13.88	11.60	−0.34	−0.64	1.00	0.94	−0.07	26.36	879
PW-BoW	91.51	11.59	2.24	4.21	1.15	0.94	0.46	11.45	74
FDA-BoW	91.51	11.59	2.24	4.21	1.15	0.94	0.46	11.45	74
SVM-BoW	16.26	11.60	0.40	0.75	1.03	0.94	0.08	19.63	371
PW-NG	91.51	11.59	2.24	4.21	1.15	0.94	0.46	11.45	74
FDA-NG	91.51	11.59	2.24	4.21	1.15	0.94	0.46	11.45	74
SVM-NG	−7.47	11.60	−0.18	−0.34	1.02	0.94	−0.04	33.98	738
PW-TDNG	62.17	11.59	1.52	2.86	1.12	0.94	0.31	15.87	166
FDA-TDNG	62.17	11.59	1.52	2.86	1.12	0.94	0.31	15.87	166
SVM-TDNG	30.65	11.60	0.75	1.41	1.02	0.94	0.15	17.92	326
S-BoW	98.87	11.79	2.22	4.32	1.15	1.00	0.99	10.79	57
S-NG	105.96	11.81	2.38	4.62	1.16	1.00	1.07	11.24	66
S-TDBG-50	104.27	11.81	2.34	4.55	1.16	1.00	1.02	11.18	58

Table 6 EUR: trading performance on the test set S_{Te}

	EUR								
	CR*	V*	SR	t-Stat	GL	ACR	PPT*	MD*	MDD
Naive-M	−84.88	10.46	−2.13	−4.02	0.89	1.00	−0.37	86.60	957
Naive-C	84.88	10.46	2.13	4.02	1.12	1.00	0.37	10.20	63
PW (cont.)	35.57	10.19	0.98	1.85	1.10	0.95	0.17	13.51	262
FDA (cont.)	36.36	10.19	1.00	1.89	1.11	0.95	0.18	14.23	261
SVM (cont.)	−21.92	10.19	−0.60	−1.14	0.99	0.95	−0.11	41.47	783
PW-BoW	58.80	10.19	1.62	3.06	1.11	0.95	0.29	11.25	115
FDA-BoW	50.33	10.19	1.39	2.62	1.11	0.95	0.25	11.25	115
SVM-BoW	−14.03	10.19	−0.39	−0.73	1.00	0.95	−0.07	35.61	887
PW-NG	3.08	10.19	0.09	0.16	1.07	0.95	0.02	29.35	602
FDA-NG	3.08	10.19	0.09	0.16	1.07	0.95	0.02	29.35	602
SVM-NG	23.88	10.19	0.66	1.24	1.02	0.95	0.12	12.30	209
PW-TDNG	23.42	10.19	0.65	1.22	1.09	0.95	0.11	23.10	326
FDA-TDNG	23.42	10.19	0.65	1.22	1.09	0.95	0.11	23.10	326
SVM-TDNG	10.80	10.19	0.30	0.56	1.01	0.95	0.05	14.51	338
S-BoW	78.83	10.45	1.98	3.87	1.11	1.00	0.67	11.24	114
S-NG	52.15	10.47	1.31	2.55	1.11	1.00	0.77	12.33	252
S-TDNG-35	11.71	10.47	0.29	0.57	1.09	1.00	0.26	18.85	690

Table 7 GBP: trading performance on the test set S_{Te}

	GBP								
	CR*	V*	SR	t-Stat	GL	ACR	PPT*	MD*	MDD
Naive-M	−89.69	9.83	−2.40	−4.52	0.90	1.00	−0.39	92.82	952
Naive-C	89.69	9.83	2.40	4.52	1.12	1.00	0.39	6.74	99
PW (cont.)	39.53	9.26	1.20	2.26	1.08	0.95	0.19	9.05	355
FDA (cont.)	44.28	9.26	1.34	2.53	1.08	0.95	0.22	6.49	109
SVM (cont.)	−6.47	9.26	−0.20	−0.37	1.01	0.95	−0.03	27.19	766
PW-BoW	86.72	9.25	2.63	4.97	1.12	0.95	0.42	5.40	55
FDA-BoW	86.72	9.25	2.63	4.97	1.12	0.95	0.42	5.40	55
SVM-BoW	3.42	9.26	0.10	0.20	1.01	0.95	0.02	22.87	368
PW-NG	86.72	9.25	2.63	4.97	1.12	0.95	0.42	5.40	55
FDA-NG	86.72	9.25	2.63	4.97	1.12	0.95	0.42	5.40	55
SVM-NG	4.87	9.26	0.15	0.28	1.00	0.95	0.02	12.74	541
PW-TDNG	79.01	9.25	2.40	4.52	1.11	0.95	0.39	5.68	115
FDA-TDNG	79.01	9.25	2.40	4.52	1.11	0.95	0.39	5.68	115
SVM-TDNG	7.64	9.26	0.23	0.44	1.02	0.95	0.04	13.46	611
S-BoW	47.21	9.74	1.27	2.48	1.09	0.99	0.60	12.11	307
S-NG	85.53	9.83	2.28	4.46	1.11	1.00	0.98	6.00	68
S-TDNG-35	77.06	9.83	2.06	4.02	1.11	1.00	0.94	8.32	102

References

Armstrong, J.S., Adya, M., & Collopy, F. (2001). *Rule-based forecasting: using judgment in time-series extrapolation*. Norwell: Kluwer.

Boser, B.E., Guyon, I.M., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on computational learning theory*, *9*, 144–152.

Cao, L.J., & Tay, F.E.H. (2003). Support vector machine with adaptive parameters in financial time series forecasting. *Neural Networks*, *14*, 1506–1518.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Dunis, C.L., & Williams, M. (2003). Application of advanced regression analysis for trading and investment. In Laws, J., Dunis, C.L., & Naïm, P. (Eds.) *Applied quantitative methods for trading and investment*. New York: Wiley.

Fisher, R.A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, *7*(2), 179–188.

Fu, T. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence*, *24*, 164–181.

Hossain, A., & Nasser, M. (2011). Reccurent support and relevance vector machines based model with applications to forecasting volatility of financial returns. *Journal of Intelligent Learning Systems and Applications*, *3*, 230–241.

Khan, A.I. (2011). Financial volatility forecasting by nonlinear support vector machine heterogeneous autoregressive model: evidence from NIKKEI225 stock index. *International Journal of Economics and Finance*, *3*, 138–150.

Kim, K.J. (2003). Financial time series forecasting using support vector machines. *Neurocomputing*, *55*, 307–319.

Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003). A symbolic representation of time series with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery (DMKD '03)* (pp. 2–11). New York: ACM.

Liu, B., Wang, X., Lin, L., Dong, Q., & Wang, X. (2008). A discriminative method for protein remote homology detection and fold recognition combining top-n-grams and latent semantic analysis. *BMC Bioinformatics*, *9*, 510.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *The Journal of Machine Learning Research*, *2*, 419–444.

Mika, S. et al. (1999). Fisher discriminant analysis with kernels. IEEE.

Mwamba, J.M. (2011). Modelling stock price behaviour: the kernel approach. *Journal of Economics and International Finance*, *3*, 418–423.

Nadaraya, E.A. (1964). On estimating regression. *Theory of Probability and its Applications*, *9*(1), 141–142.

Ou, P., & Wang, H. (2010). Financial volatility forecasting by least square support vector machine based on GARCH, EGARCH and GJR models: evidence from ASEAN stock markets. *International Journal of Economics and Finance*, *2*, 51–64.

Perez-Cruz, F., Afonso-Rodriguez, J.A., & Giner, J. (2003). Estimating GARCH models using support vector machines. *Quantitative Finance*, *3*, 163–172.

Schölkopf, B., & Smola, A.J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Cambridge: MIT Press.

Sharpe, W.F. (1966). Mutual fund performance. *Journal of Business. Series A*, *39*, 119–138.

Shawe-Taylor, J., & Cristianini, N. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge: Cambridge University Press.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis infrastructure*. Cambridge: Cambridge University Press.

Smola, A.J. (2007). An introduction to machine learning: instance based estimation. Statistical machine learning program. Lecture notes. Tata Institute, Pune. http://alex.smola.org/teaching/pune2007/pune_2.pdf.

Tay, F.E.H., & Cao, L. (2001). Application of support vector machines in financial time series forecasting. *Omega*, *29*, 309–317.

Van Gestel, T., Suykens, J.A.K., Baestaens, D.E., Lambrechts, A., Lanckriet, G., Vandaele, B., De Moor, B., & Vandewalle, J. (2001). Financial time series prediction using least squares support vector machines within the evidence framework. *Neural Networks*, *12*, 809–821.

Vapnik, V. (2013). The nature of statistical learning theory, Springer Science & Business Media.

Watson, G.S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics. Series A*, *26*(4), 359–372.